2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing

SmartProbe: a Bottleneck Capacity Estimation Tool for Smartphones

Francesco Disperati*, Dario Grassini*, Enrico Gregori†, Alessandro Improta†,
Luciano Lenzini*, Davide Pellegrino*, Nilo Redini*
*Information Engineering Department, University of Pisa, Pisa, Italy
l.lenzini@iet.unipi.it
{f.disperati — d.grassini — d.pellegrino — n.redini}@studenti.unipi.it
†Institute of Informatics and Telematics, Italian National Research Council, Pisa, Italy
{alessandro.improta — enrico.gregori}@iit.cnr.it

***Abstract*—The recent development of Internet cellular and wireless networks led several mobile phone manufacturers to create smartphones able to connect to the Internet anytime-anywhere. Despite the growing dependency of mobile phones from the Internet, very little efforts have been done to measure wireless network performances from smartphones. In this paper, we introduce SmartProbe, a tool dedicated to smartphones able to discover the bottleneck link capacity between two hosts. This tool is based on the technique proposed in PBProbe, but uses a lower amount of resources, making it appealing to the smartphone environment, where the battery consumption and costs are topics of extreme importance. Thanks to the capability of smartphones to understand the type of network where they are connected and to the technique exploited to choose the correct size of the probing packet train, Smartprobe is able to save on average more than 80% of data if compared to the original technique depicted in PBProbe, still maintaining a comparable precision.**

***Keywords*-Capacity Estimation, Internet, Smartphones**

I. Introduction

The first model of mobile phone was developed in the far 1973 by Martin Cooper. Since then, the mobile phones slowly entered in the life of everyone, especially during the last years thanks to the production of innovative mass-directed smartphones. The main reason of the large success of smartphones is due to the development of new high-speed access networks by mobile phone operators that allowed them to access the Internet services, such as world wide web and e-mail and, as consequence, allowed the mobile users to be connected anytime-anywhere. This represented a huge improvement in the mobile phone evolution and gave birth to a whole new set of economic opportunities for service developers that, in these days, are experiencing the same boom of work requests experienced during the late 1990s-early 2000s by web developers. The linkage between smartphones and Internet is tightening even more as time advances. The quality of the Internet connection used by the smartphone has thus gained a fundamental role in the quality of the services offered by the smartphone itself. Nowadays a smartphone is typically connected to the Internet via Wi-Fi 802.11 a/b/g/n whenever an access point is available (e.g. at home or workplaces), or via 2G/3G/4G connections offered by mobile telecommunication companies. Nevertheless, the number of user-available applications developed to check the quality of Internet connection from smartphones is very small, and often it is not clear how data is collected. In this paper we exploit the large amount of literature available on the analysis of the link characteristics both in wired and wireless environments in order to create an innovative tool for smartphones, named *SmartProbe*, which allows users to understand the effective quality of service that they are experiencing with each other or with their service providers. To achieve that, we propose a tool able to estimate the bottleneck capacity – that in this paper is considered to be the maximum achievable throughput of the narrowest link of the probed path [1] – between two end users. To do that, we enhanced the packet train based tool PBProbe [2], minimizing the amount of data sent and, consequently, the energy consumption of the smartphones. The techniques that we devised lead to a considerable amount of data saved in comparison with PBProbe, reaching peaks of 96% in the case of 3G networks and peaks of about 89% in Wi-Fi networks.

The paper is organized as follows. Sect. II provides an overview of the tools available in literature. Sect. III briefly describes the techniques introduced by PBProbe and highlights the reasons that makes PBProbe unusable on smartphones. Sect. IV describes SmartProbe, with particular regards to the techniques aimed to minimize its costs. Sect. V analyzes its performances and compares the costs of PBProbe and SmartProbe. Finally, Sect. VI concludes the paper.

II. Related work

The first tool focusing on the discovery of the bottleneck capacity of Internet links was developed by Van Jacobson in 1997. Since then, a plethora of tools have been developed for wired networks, typically based on packet pairs (e.g. *Bprobe* [3], *Pathrate* [4], *CapProbe* [5] and *TOPP* [6]) and packet trains (e.g. *PBProbe* [2] and *Cprobe* [3]). Since these tools are not conceived for wireless environments, their inferences can be inaccurate. Recently, some dedicated tools based on packet pairs have been developed to infer the bottleneck capacity of a wireless link (e.g. *WBest* [7] and *AdHoc-Probe* [8]), but suffer heavily of compression and expansion phenomena experienced by the packet pairs that are caused by media access control and contending traffic [9] and by Interrupt Coalescence phenomena [10]. Packet train tools are able to minimize the effects of the

Interrupt Coalescence, but they still suffer of contending traffic interferences. Among all these techniques only PBProbe was claimed to work fine on wireless networks [2]. Nevertheless, it is not possible to use it as it is on smartphone environments due to the excessive required data and, consequently, due to the excessive resource consumption. To the best of our knowledge, this work is one of the first efforts towards adapting the well-known bottleneck link capacity discovery techniques to work efficiently on smartphones. The only tools for smartphones implemented so far are dedicated to the discovery of the bulk transfer capacity [11], i.e. the amount of data that can be sent between two ends via TCP, that is something different from the bottleneck capacity [1].

III. Towards a smartphone-dedicated capacity estimation tool

PBProbe [2] is one of the first capacity estimation techniques based on packet trains instead of relying on packet pairs, which were exploited in the past by a large set of tools such as *Pathrate* [12] and *CapProbe* [5]. The rationale behind this choice relies on the inaccuracies that packet pair techniques suffer in presence of high speed links [2] and the Interrupt Coalescence [10]. This because they estimate the capacity as $C = \frac{P}{T}$, where P is the size of the packet and T is the dispersion among the pair of packets sent. In high speed networks, C is large, P is limited by the MTU value of the link and, as consequence, T gets very small. This leads to time resolutions problems on the hosts. For example, to correctly compute the capacity of a bottleneck link in a common 1 Gbps network, $T \simeq 110\mu s$. In presence of the Interrupt Coalescence – a technique often used to limit the system overhead introduced by the interrupt rate generated by the Network Interface Cards (NICs) in receiving packets in high speed networks – T is compressed due to data buffering on the NIC, and the resulting C can be misleading. Given these problems, it is easy to understand that the estimation process of tools based on packet pairs is also very sensitive to external noises introduced either by cross-traffic and by the process scheduler of the host. These problems are solved by enlarging the numerator of previous equation by sending a larger amount of data from source to destination. Since the MTU cannot be enlarged indefinitely, the only possibility is to send a larger number of packets. Note that it has been proved that using packets as large as possible reduces the effect of queueing delay noise and of timestamp resolution at the receiver side [12]. The capacity can thus be computed as:

$$C = \frac{kP}{D} \qquad (1)$$

where k is the size of the train and D is the dispersion between the first and last packet of the train. Note that we consider that $k+1$ packets are sent back-to-back from source to destination when a train length is equal to k in order to not cause confusion between this work and PBProbe [2].

Another key feature of PBProbe is the usage of the *delay sum* – firstly introduced in CapProbe [5] – to minimize the under- and over-estimations due to queue delays caused by

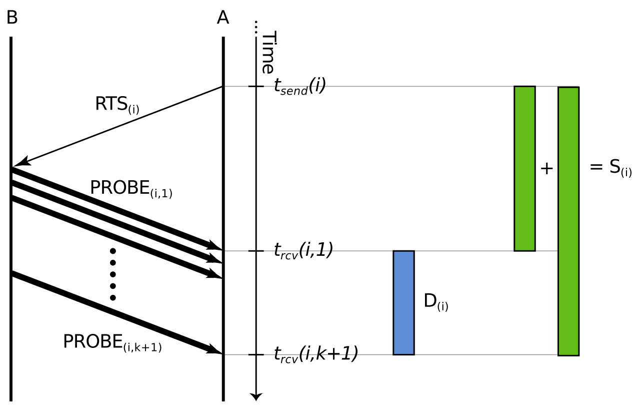

Fig. 1: Delay sum components

cross-traffic. As shown in Fig. 1, the delay sum S_i is computed as the sum of the delays experienced by the first and the last packet and is computed for each train i. Thus, in a campaign of n trains, the minimum delay sum identifies the train m which has experienced less queue delay effects. Once identified the best train in a campaign, the capacity is computed with equation 1 using the dispersion D_m experienced by train m.

PBProbe was originally conceived to estimate the bottleneck capacity of high-speed wired networks, but it was also proved to correctly infer the bottleneck capacity in wireless environments [2]. The main problem that do not allow a direct application of PBProbe on mobile phones concerns with its resource consumption. To obtain a valid result, PBProbe uses $n = 200$ trains composed each by k packets of 1,500 bytes. The larger the value of k is, the larger is also the dispersion time D experienced between the arrival of the first and last packet of each train and, as a consequence, the smaller is also the impact of potential interference phenomena caused by the Interrupt Coalescence [10] and by timer resolution limitations. To identify the correct value of k, a dynamic algorithm which compares the dispersion time D experienced with a threshold value D_{thresh} (this value depends on the system timer resolution) is applied. In detail, if any of the trains experience $D \leq D_{thresh}$, then the train length k is increased by ten-fold and the computation starts again from scratch. If we consider $D_{thresh} = 1ms$ as a good threshold value to limit the impact of system interferences, as assumed in [2], this means that PBProbe would require about two thousand packets, i.e. about 3MB of data, to compute correctly the capacity of a common 802.11g wireless network. These values increase tenfold if we consider $D_{thresh} = 10ms$, which is a

Network type		**Nominal Capacity (Mbps)**	**Dispersion (ms)**		
			k = 1	**k = 10**	**k = 100**
WiFi 802.11	*b*	11	1.09	10.91	109.09
	a,g	54	0.22	2.22	22.22
	n	450	0.03	0.27	2.67
Mobile 2G	*GPRS*	0.171	70.18	701.75	7,017.54
	EDGE	0.473	25.37	253.7	2,537
Mobile 3G	*UMTS*	1.8	6.67	66.67	666.67
	HSPA	14.4	0.83	8.33	83.33
Mobile 4G	*LTE*	326.4	0.04	0.37	3.68

Table I: Minimum train length set by PBProbe algorithm

more realistic value for devices with bounded resources like smartphones. In this case, PBProbe would require more than twenty thousand packets, i.e. about 30MB of data. In Table I are listed the dispersion values that would be experienced by PBProbe with a variable value of k on wireless and mobile networks, computed by considering the nominal capacity values of each network. Note that the minimum number of packets required by PBProbe to correctly infer the bottleneck capacity of mobile networks ($D_{thresh} = 10ms$) is between about 300KB and 300MB of data. This often represents a technical issue, since a high amount of transferred data is likely to conduct to a significant amount of consumed energy of the mobile devices and, depending on the commercial agreement between user and mobile operator, also to high economic costs.

IV. SmartProbe: PBProbe over smartphones

The excessive energy consumption and the potential costs for users are the main reasons that made us to develop *SmartProbe*, an enhanced version of PBProbe dedicated to smartphones and tailored on wireless and mobile networks. Differently from PBProbe, SmartProbe takes heavily into account the limits of the common smartphones and infers the capacity of the bottleneck link by using only a small amount of data and limiting the impact of the tool on the performances perceived by the users. In this section we provide a brief description of the tool and of its protocol, and we analyze in detail the enhancements applied to the original tool in order to better fit in the smartphone environment.

A. Protocol overview

The SmartProbe protocol (Fig. 2) is similar to PBProbe and is composed by two symmetrical phases in which the downlink and the uplink bottleneck capacities for each host are respectively estimated. Initially, one of the two hosts is considered as the *Estimator* (E) – i.e. the host which asks for packets and computes the capacity estimation – while the other is considered to be the *Prober* (P), i.e. the host that waits for requests and effectively send packet trains. The two phases consist in an UDP handshake between Prober and Estimator to synchronize and initialize the two hosts, followed by the effective transmission of the train of UDP packets. In detail, the Estimator starts the handshake by sending a `START` message to the Prober, that replies with an `ACK` message, communicating its effective presence to the Estimator. After that, each host calculates the minimum value k required by each host to compute a value of capacity without being affected by system interferences, as will be introduced in Section IV-B. Then, the smaller among the two values is chosen for the experiment, since the slower link should introduce a delay large enough for the smartphone connected on the faster link to compute a correct capacity value. Then, the Estimator proceeds to start the measurement campaign by sending a `RTS` (Request To Send) UDP message, which triggers the dispatch of a UDP train. Note that the `RTS` message contains the number of packets k that the Prober has to use in each train, and that the dispatch of every UDP control message also set a timeout

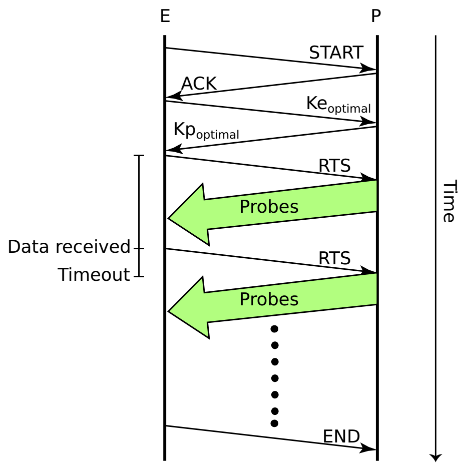

Fig. 2: SmartProbe protocol

which allows the hosts to not stale in case of packet losses. Once the Estimator receives correctly all the packets of a train, then it computes the relative delay sum S_i and dispersion D_i, otherwise the train is invalidated at the timeout expiration. If more than three failures are experienced, the hosts assume that the network is congested and the experiment is restarted by halving the train length. The computation is considered to be completed when n valid experiments are achieved. Then, in order to terminate the estimation, an `END` message from the *Estimator* to the *Prober* is sent. Finally, the bottleneck capacity value is calculated by applying equation 1 selecting the dispersion value related to the lowest delay sum S_m among the collected samples. Once the first computation is completed, Prober and Estimator exchange their roles and the procedure is repeated, obtaining an estimation of the bottleneck capacity both in uplink and downlink for each of the hosts.

B. SmartProbe energy-saving features

SmartProbe takes cue from the technique described in [2] by improving some of its features leveraging energy- and cost-saving rationales, in order to use the smallest amount of resources to achieve a correct result and, at the same time, to not drain the battery of the smartphones. Here in the following are described the rationale behind these enhancements in detail.

1) Packet train length k: The choice of a proper value k is one of the main differences between PBProbe and SmartProbe. In PBProbe, the value of k is chosen dynamically by analyzing the time dispersion and increasing it ten-fold each time a train is received with a dispersion value D lower than a preset dispersion threshold D_{thresh}. As already shown in Section III (Table I), this can lead to an excessive and unnecessary amount of traffic on smartphones. The SmartProbe approach is exactly the opposite of the PBProbe choice and exploits the a priori knowledge about the type of network to which the hosts are connected – which can be extracted directly from the smartphone operating system – and about the nominal capacity

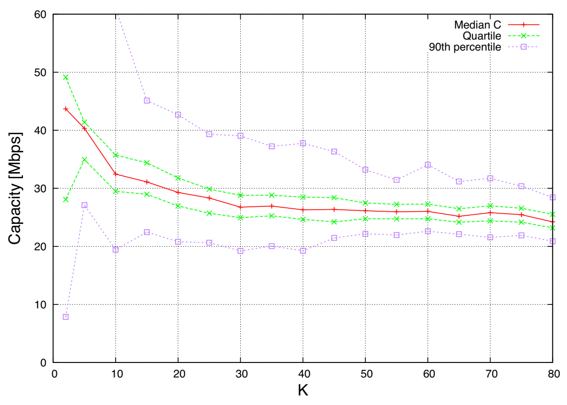

Fig. 3: Dispersion experiment results

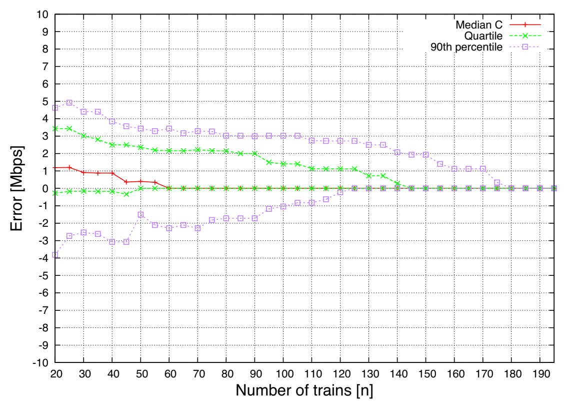

Fig. 4: Train length experiment results

of that type of network, in order to identify an ideal value $k_{optimal}$ which should be used in the estimation of C (see Equation 1). The nominal capacity typically represents an upper bound of the real wireless environments [13], thus it can be used to understand the minimum number of packets required to experience a dispersion value larger than D_{thresh}. The value of k then can thus be dynamically lowered by the application whenever one of the hosts receive too many trains experiencing packet loss. This can be caused for example by the presence of traffic shapers on the link or by excessive contending traffic on the wireless access point. In those cases, the application still tries to retrieve a correct value of bottleneck capacity by lowering the value of k up to 1, i.e. using a simple packet pair.

In PBProbe, D_{thresh} was set to $1ms$ basing on experimental results obtained from wired servers, which have quite a different amount of resources from smartphones. To understand the ideal value of D_{thresh} on smartphones, we performed an analysis on the behavior of SmartProbe on an unloaded 802.11g network by manually imposing the value of k and running 100 measurement campaigns for each value of k chosen. The results are depicted in Fig. 3. As can be seen, the average results of the tool converge to a defined capacity value by using trains composed by more than *forty* packets. These values of k, by applying equation 1, lead to a dispersion value $D \sim 10ms$, which we consider to be the minimum dispersion value D_{thresh} such that the effects of operating system interferences are minimized in the estimation of C. Thus, it is possible to obtain the ideal value of k from which the computation shall start by applying equation 1, and considering C as the nominal capacity in each network and D as D_{thresh}. These values can be used then on real experiments, since the dispersions collected on real wireless networks will be surely larger than the ideal case. Table II summarizes the values of k found for each wireless network analyzed: the faster the network is, the larger is the value required. The SmartProbe algorithm is described in detail in Fig. 5.

2) Number of train per campaign n: Another factor that makes PBProbe unusable on smartphones is that it uses a fixed amount of 200 trains for each measurement campaign, that was a value found to be valid by the same authors of PBProbe in [5] and [14]. In order to decrease the amount of data sent – and therefore the costs in terms of connection fee and smartphone battery consumption – n has to be lowered consistently. To find a proper value of n we performed a campaign of tests consisting in 100 experiments with $n = 200$, by connecting a smartphone to an 802.11g wireless network and connecting a server on the same wired LAN of the wireless access point. In each test we firstly retrieved the most reliable capacity C_{200} by using the full sample list like in PBProbe computation. Then, we compute the capacity C_m that would be retrieved by considering only the first m trains and we compute the relative error between C_{200} and C_m. Fig. 4 shows the distribution of the medians of the relative errors computed for each m, as well as its inter-quartile and 90th percentile ranges. As we can see, the median value of the errors is close to zero after $m = 60$, meaning that $n = m = 60$ is enough to infer the correct value of capacity most of the times. Obviously, the decrease of the amount of attempts performed leads to an unavoidable leakage of accuracy. However in an environment where energy-saving is fundamental, the choice of $n = 60$ represents a good trade-off between accuracy and performances. This value is valid also for other wireless network technologies.

Network		**k**
WiFi 802.11	*b*	8
	a,g	44
	n	374
Mobile 2G	*GPRS*	1
	EDGE	1
Mobile 3G	*UMTS*	1
	HSPA	11
Mobile 4G	*LTE*	271

Table II: Train size k per wireless technology

```
Send START packet
i ← 0
failed ← 0
$D_{min} \leftarrow \inf$
$S_{min} \leftarrow \inf$
$k \leftarrow k_{optimal}$
while(i $< n$)
    Set Timeout
    Receive packet train $i$
    if (Timeout triggered)
        failed ← failed+1
        if ( failed == 3)
            i ← 0
            k ← k/2
            $D_{min} \leftarrow inf$
            $S_{min} \leftarrow inf$
            continue
    else
        Measure $D_i$ and $S_i$
        if ($S_i < S_{min}$)
            $S_{min} \leftarrow S_i$
            $D_{min} \leftarrow D_i$
    i ← i+1
Calculate capacity with $D_{min}$
Send END packet
```

Fig. 5: SmartProbe estimation algorithm

3) Timeout value: PBProbe uses an adaptive timeout in order to achieve a good trade-off between algorithm efficiency and link load balancing, in order to not overload the narrow link. Specifically, the value of the timeout in PBProbe is calculated as $G = 2 \cdot D_{min} - (t2 - t1)$ where $(t2 - t1)$ represents the time elapsed between sending the RTS packet and receiving the last packet of a packets train, and D_{min} is the minimum value of dispersion experienced during the algorithm. Hence a new packet train is requested each G seconds. In a wired environment this is tolerable since the dispersions values inside an experiment are close to each other, but this is no longer valid in a wireless environment. In wireless experiments two consecutive trains may experience extremely different dispersions due to the high link variability. For example, in 802.11 different SNR may lead to a different transmission rate, and thus to different measured values of capacity. Using the adaptive timeout suggested by PBProbe, this could lead to a high amount of valid but slow attempts that are discarded, with the result of requesting additional unrequired packet trains, thus increasing the data consumption. Moreover, introducing a gap between two consecutive trains may lead to trigger energy-saving mechanisms associated to the wireless technology used [15], leading to unwanted delays and, thus, to measurement errors. In SmartProbe we solved this problem by using a fixed value for the timeout large enough to tolerate the wireless link variability, and by imposing an immediate transmission of the following train as soon as an

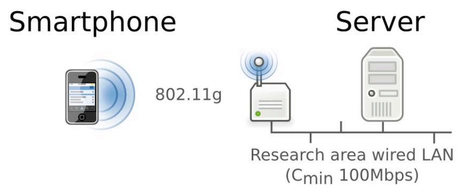

Fig. 6: IIT-CNR Wireless scenario

Network		PBProbe [MB]	Smartprobe [MB]	Smartprobe worst case [MB]
WiFi 802.11	*b*	~ 3	~ 0.7 (-77%)	~ 1.7 (-44%)
	a,g	~ 30	~ 4 (-87%)	~ 8.2 (-73%)
	n	~ 300	~ 33.7 (-89%)	~ 67.6 (-78%)
Mobile 2G	*GPRS*	~ 0.6	~ 0.2 (-67%)	~ 0.2 (-67%)
	EDGE	~ 0.6	~ 0.2 (-67%)	~ 0.2 (-67%)
Mobile 3G	*UMTS*	~ 3	~ 0.2 (-93%)	~ 0.2 (-93%)
	HSPA	~ 30	~ 1 (-96%)	~ 2 (-94%)
Mobile 4G	*LTE*	~ 300	~ 24 (-92%)	~ 48 (-84%)

Table III: PBProbe vs SmartProbe data usage comparison ($packet\_size = 1500B$, $total\_size = packet\_size * K$)

attempt is completed. One attempt is repeated only if the timeout triggers, since it means that the estimation is unreliable due to the presence of packet loss. A strict requirement for our timeout value is that it has to be higher or equal than $RTT + D$ considering the worst case. In order to calculate the worst case of dispersion, we considered 50 kbps as a lower bound for our estimations, since SmartProbe is designed to be used with commercial networks. Moreover, we consider $k = 1$ as the minimum amount of data required by the tool to work, i.e. a packet pair. Therefore, we achieve a dispersion value of $D = 480ms$. About the RTT worst case, we consider the case where the hosts are located the farthest as possible. As a plausible value of RTT, we used twice the time required to ping a machine in Australia from Italy (about $750ms$). In addition to that, we also modified the algorithm of PBProbe by avoiding cases of deadlocks caused by UDP packet loss: the timeout is set each time a `RTS` message is sent, it is reset each time a train is correctly received and the attempt is performed once again whenever the timeout triggers.

V. Results

Since SmartProbe is an enhancement of PBProbe on wireless networks, it is straightforward that the most interesting analysis regards a comparison between the data usage required by the the two tools. Considering a $D_{thresh} = 10ms$, PBProbe would require trains of 100 packets to estimate the capacity of a 802.11g network according to Table I. Thus, a PBProbe

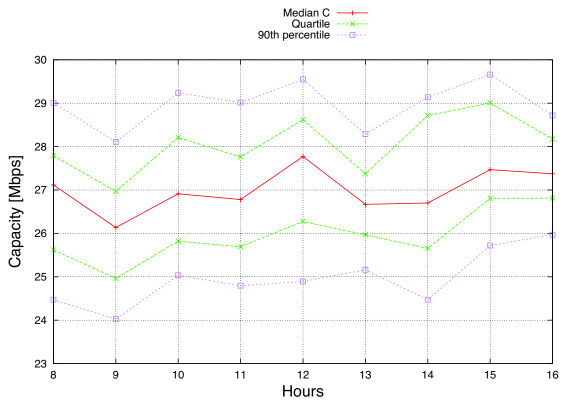

Fig. 7: Bottleneck capacity experienced by smartphone in IIT-CNR Wireless scenario

estimation requires $n = 200$ trains with $k = 100$, for a total of about 30 MB of data. SmartProbe, on the other hand, starts the estimation using a value $n = 60$ and $k = 44$ (see Table II) for a total of about 3.9 MB, about the 13% of data used by PBProbe. On the other hand, by considering the worst case in which a persistent packet loss causes the length of packet trains to decrease to the smaller value, i.e. $k = 1$, SmartProbe would require 8.19 MB of data[1], still the 27% of data sent by PBProbe in the ideal case. Results for other types of wireless networks can be found in Table III. By sending a smaller amount of data, SmartProbe obtains obvious benefits in terms of battery consumption and user economic costs.

SmartProbe has also proved to be able to retrieve good results from real world wireless environments. As a significative case study, we performed an analysis on the wireless network used commonly during work hours by 20-30 researchers of IIT-CNR in Pisa. The test scenario is depicted in Fig. 6. The server is an Intel(R) Pentium(R) 4 CPU 2.80GHz machine with 2 Gb RAM running Linux Ubuntu 11.04 Natty Narwhal and is running a Java version of SmartProbe, while the smartphone is a Samsung Galaxy Nexus GT-19250 running Android 4.0.3 ICS (Ice Cream Sandwich). Both the server and the access point are connected on the same Ethernet LAN with 100Mbps NICs, while the smartphone is connected via 802.11g to the access point. We performed 100 experiments each hour from 8AM to 4PM with $k = 50$ and $n = 60$, testing the tool on different network loads. Results experienced by the smartphone can be found in Fig. 7 in terms of median, interquartile and 90th percentile every hour. SmartProbe results are quite stable around the median value, meaning that SmartProbe is not strongly affected by the different volumes of traffic generated at different working hours.

VI. Conclusions

In this paper we described SmartProbe, an enhanced version of PBProbe tailored for smartphones. SmartProbe is designed to be used on smartphones and devices with an energy-saving perspective, and its choices have been aimed at optimizing parameters such as execution time and cost in the broadest sense. To the best of our knowledge, in the current state of the art does not exists any application that – in addition to accurately measure the bottleneck link capacity by minimizing the effect of smartphone operating system interferences – pays attention to reducing the amount of generated traffic and, consequently, the energy consumption. Thanks to its ability to adapt itself to the type of wireless networks and to the choices made on the size of the trains and on the number of trains used, the tool is able to guarantee together with a high precision even an upper bound in terms of amount of data sent, which has been proved to be markedly better than those which could provide PBProbe in its optimal case. Note also that, being based on PBProbe, our methodology also works on the wired environments.

Smartprobe provides to end users a useful tool to understand the quality of their Internet connections. Wireless links and home xDSL connections typically represent the *narrowest* link in terms of capacity in any Internet path. With Smartprobe it is possible to retrieve the effective service quality offered by their service providers, given that one of the two hosts is located on a network that cannot be the bottleneck of the tested path. In this perspective, we plan to introduce Smartprobe in the Portolan infrastructure [16] in the very next future. Portolan is a crowdsourcing-based system which aims to discover from smartphones the Internet topology together with its characteristics. We plan to introduce in the Portolan architecture a high-performance server connected via Gigabit Ethernet to the Internet and which is intended to be one of the two ends of Smartprobe. We also plan to improve the Capacity estimation accuracy by introducing a preliminary phase dedicated to contention-based media access wireless protocols (e.g. Wireless LANs) [17].

[1]In the worst case, SmartProbe would send firstly $n = 60$ trains with $k = 44$, then $n = 60$ trains with $k = 22$, and so on with $k = 11, 5, 2, 1$, leading to a total of 5,460 packets, i.e. 8.19 MB of data

References

[1] C. Dovrolis, R. Prasad, M. Murray, and K. C. Claffy, "Bandwidth estimation: metrics, measurement techniques, and tools," *IEEE Network*, vol. 17, no. 6, pp. 27–35, Apr 2003.

[2] L.-J. Chen, T. Sun, B.-C. Wang, M. Y. Sanadidi, and M. Gerla, "PBProbe: A Capacity Estimation Tool for High Speed Networks," *Comput. Commun.*, vol. 31, no. 17, pp. 3883–3893, 2008.

[3] R. L. Carter and M. E. Crovella, "Measuring bottleneck link speed in packet-switched networks," *Performance Evaluation*, vol. 27-28, pp. 297–318, Oct 1996.

[4] C. Dovrolis, P. Ramanathan, and D. Moore, "Packet-dispersion techniques and a capacity-estimation methodology," *IEEE/ACM Transaction on Networking*, vol. 12, no. 6, pp. 963–977, Dec 2004.

[5] R. Kapoor, L.-J. Chen, L. Lao, M. Gerla, and M. Y. Sanadidi, "CapProbe: a Simple and Accurate Capacity Estimation Technique," in *Proceedings of the ACM SIGCOMM '04*, 2004, pp. 67–78.

[6] B. Melander, M. Bjorkman, and P. Gunningberg, "A new end-to-end probing and analysis method for estimating bandwidth bottlenecks," in *Proceedings of GLOBECOM '00*, vol. 1, 2000, pp. 415–420.

[7] M. Li, M. Claypool, and R. Kinicki, "Wbest: a bandwidth estimation tool for ieee 802.11 wireless networks," in *Proceedings of Local Computer Networks (LCN) '08*, 2003, pp. 39–44.

[8] L.-J. Chen, T. Sun, G. Yang, M. Y. Sanadidi, and M. Gerla, "Adhoc probe: End-to-end capacity probing in wireless ad hoc networks," *Wireless Networks*, vol. 15, no. 1, pp. 111–126, Jan 2009.

[9] M. Li, M. Claypool, and R. Kinicki, "Packet dispersion in ieee 802.11 wireless networks," in *Proceedings of Local Computer Networks (LCN) '06*, 2006, pp. 721–729.

[10] R. Prasad, M. Jain, and C. Dovrolis, "Effects of interrupt coalescence on network measurements," in *Proceedings of Passive and Active Network Measurement (PAM) Workshop '04*, vol. 3015, 2004, pp. 247–256.

[11] S. Bauer and D. Clarke, "Understanding broadband speed measurements," *38th Research Conference on Communication, Information and Internet Policy*, Sep 2011.

[12] C. Dovrolis, P. Ramanathan, and D. Moore, "What Do Packet Dispersion Techniques Measure?" in *Proceedings of the IEEE INFOCOM '01*, 2001, pp. 905–914.

[13] F. Cal, M. Conti, and E. Gregori, "Dynamic tuning of the ieee 802.11 protocol to achieve a theoretical throughput limit," *IEEE/ACM Transactions on Networking*, vol. 8, pp. 785–799, 2000.

[14] L.-J. Chen, T. Sun, G. Yang, M. Y. Sanadidi, and M. Gerla, "Monitoring access link capacity using tfrc probe," *Computer Communications*, vol. 29, no. 10, pp. 1605–1613, Jun 2006.

[15] "IEEE Draft Standard for Local and Metropolitan Area Networks - Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE P802.11ad/D8.0, May 2012 (Draft Amendment based on IEEE 802.11-2012)*, pp. 1–667, 2012.

[16] "University of Pisa Portolan project," http://portolan.iet.unipi.it.

[17] P. Kanuparthy, C. Dovrolis, K. Papagiannaki, S. Seshan, and P. Steenkiste, "Can User-level Probing Detect and Diagnose Common Home-WLAN Pathologies," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 1, pp. 7–15, 2012.