QProbe: Locating the Bottleneck in Cellular Communication

Nimantha Baranasuriya†, Vishnu Navda‡, Venkata N. Padmanabhan‡, Seth Gilbert†
†National University of Singapore ‡Microsoft Research India

ABSTRACT

Mobile communication is often frustratingly slow. When a user encounters poor performance, and perhaps even "confirms" the same by running a speed test, the tendency is to ascribe blame to the user's last-mile provider. However, as we argue in this paper, a more nuanced approach is needed to identify the location of the bottleneck responsible for the poor performance. Specifically, we focus on the question of whether the bottleneck lies in the cellular last hop (3G or LTE link) or elsewhere in the WAN path.

We present `QProbe`, a tool that takes advantage of the proportional fair (PF) scheduler employed in cellular networks to determine whether queuing is happening at the cellular link. After validating `QProbe` through simulations and controlled experiments, we present our findings from a measurement study conducted over a 2 month period involving over 600 real-world users across 51 operator networks in 33 countries. We find that, for example, the cellular last-hop link is the bottleneck in 68.9% and 25.7% of the total bottleneck cases for 3G and LTE clients, respectively, suggesting that there is a significant fraction of cases where the poor performance experienced by the user is due to the WAN and could potentially be routed around. Moreover, we show that `QProbe` detects the bottleneck link location with greater than 85% accuracy for both 3G and LTE clients in our measurement study.

CCS Concepts

•**Networks** → **Application layer protocols; Mobile networks; Network measurement;**

Keywords

Cellular Networks; LTE; 3G; Bottleneck Detection

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CoNEXT '15, December 01-04, 2015, Heidelberg, Germany
© 2015 ACM. ISBN 978-1-4503-3412-9/15/12...$15.00
DOI: http://dx.doi.org/10.1145/2716281.2836118

1. INTRODUCTION

Cellular connections at times are frustratingly slow. While it might be natural for users to blame the cellular link for the poor performance, doing so is not always appropriate. Indeed, modern cellular connections are often faster than WiFi (e.g., [4] reports that LTE outperforms WiFi 40% of the time). Therefore, we need a more nuanced approach to identifying the location of the bottleneck.

Broadly, we would like to know whether the bottleneck responsible for poor performance lies in the "last-mile" cellular wireless link or elsewhere in the WAN path. Knowing this is key to remediating the problem. For example, if the problem is *not* in the last-mile wireless link, one might route around the WAN bottleneck, say by picking a different replica (e.g., a different CDN server). On the other hand, if the cellular link is the bottleneck, the user may have to look for an alternate connection (e.g., WiFi) or have the application adapt (e.g., by downsizing media content). Any such adaptation undertaken by the client would be over and above any remediation (e.g., load balancing) performed by the cellular provider itself, thus having the advantage of being provider-independent.

The problem of locating the bottleneck in the context of cellular-connected clients is challenging for several reasons. First, cellular base stations employ per-station queues with proportional fair (PF) scheduling [12], unlike the FIFO queuing and servicing assumed in past work on identifying and estimating bottleneck capacity on wired Internet paths (e.g., [7]). Second, cellular data plans tend to have tight data caps, so the bottleneck detection algorithm needs to be lightweight in terms of data usage, avoiding expensive data-intensive probing. Third, cellular networks tend to be closed, under the tight control of the operator, leaving clients with little visibility into the state of the network (e.g., the network load) and no access to a vantage point within the cellular network.

We present `QProbe`, a lightweight, active probing technique to locate the bottleneck link on an end-to-end path, specifically, to decide whether the bottleneck lies on the wired WAN path or on the cellular last-mile. `QProbe` works by detecting whether there is queuing on either segment, hence its name. This detection is enabled by the very different behaviour of FIFO queuing on the wired path and PF scheduling at the cellular base station. When a train of small probe packets is sent by `QProbe` running on a remote host, these

tend to get clumped together into *back-to-back bursts* when there is queuing at the cellular base station, much more so than when there is queuing on the wired path. On the other hand, when TTL-limited (large) load packets are interspersed with the small probe packets, then the inter-packet spacing between the latter tends to get *stretched* when there is queuing on the wired path but is unaffected by queuing at the basestation. These 2 signals — back-to-back bursts and stretching — are used in combination by `QProbe` to locate the bottleneck.

We make three main contributions in this paper:

1. We have designed a novel, lightweight probing technique to disambiguate between WAN and cellular (3G /LTE) bottlenecks in time on the order 700 ms.
2. Using simulations and controlled experiments, we validate the `QProbe` technique and show that it can locate the bottleneck with high accuracy.
3. We analyze data from our deployment of `QProbe` as an iPhone app to over 600 users spread across 33 countries and 51 operators and show that `QProbe` classifies the bottleneck with greater than 85% accuracy.

2. CELLULAR SCHEDULING

In this section, we highlight the unique characteristics of the downlink packet scheduler in cellular base stations. We then provide insight on how these are leveraged in `QProbe`.

Unlike WiFi and wireline routers, cellular base stations employ a centralized *proportional fair scheduler* (or "PF-scheduler") that determines which client(s) should be served in each scheduling interval (also referred to as a time slot or TTI). Each slot is typically a few milliseconds long (2 ms in 3G). Multiple clients can be simultaneously served in a single time slot by assigning different orthogonal codes (time-frequency resource blocks) for 3G (LTE) clients.

The objective of the PF-Scheduler is to improve the aggregate network throughput, while also being fair to the clients. Instead of strict round-robin scheduling, the PF-Scheduler picks a client that maximizes the ratio of instantaneous rate to the average allocated rate for each client over a certain time window. This ensures that no client is starved, while also keeping the aggregate network throughput healthy.

As the number of clients in the network increases, the slot allocations for a client tends to get spaced apart in time, as the base station is serving other clients. However, when a client is scheduled, multiple packets buffered at the base station could well be transmitted back-to-back, depending on the bitrate associated with the instantaneous link quality and the average allocated rate from the recent past.

In order to gather slot-level data, we used the QXDM diagnostics tool from Qualcomm on a rooted Windows Phone device. We performed a bulk download over 3G on one device, adding 0, 1, or 2 other downloaders in the background. (We ran these experiments in the middle of the night to avoid interference from other users.) Figure 1(a) clearly shows that as the number of downloaders increased, the inter-slot gaps also increased, from a median of 2 ms to 6 ms with the addition of just two downloaders; the throughput also dropped correspondingly. During peak hours in crowded settings, we have seen inter-slot gaps even larger than 6 ms.

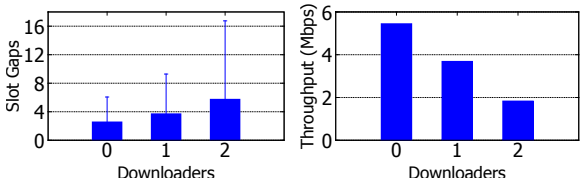

Figure 1: (a) Slot gaps between assignments, and (b) Throughputs observed by the measurement device with different number of background downloaders

Note that cross-traffic on a WAN link can also cause large gaps to be introduced between packets destined to a client. The key difference, however, is that with cellular networks, when the PF-Scheduler chooses a client for downlink transmission, multiple packets can be scheduled back-to-back, especially if the packets are small and the inter-slot gap is large. `QProbe` leverages this for its detection, especially since inter-packet gaps can easily be measured by the client.

3. QPROBE DESIGN

In this section, we begin with a description of the design constraints for the bottleneck detection algorithm. We focus on localizing the problem to one of two parts in an end-to-end path, and describe our methodology for each part separately. Finally, we put it all together and describe the `QProbe` technique that combines these design elements.

3.1 Design Requirements

Our focus in this paper is on diagnosing pathological cases when the observed throughput from an Internet server to a cellular-connected client is quite low. The measurement technique should be able to pinpoint the location of the bottleneck to one of the two segments in an end-to-end path: (a) wired WAN path; or (b) cellular wireless last hop. The wired path includes the Radio Network Controller (RNC) optical backbone within the provider's network as well as the WAN beyond. Typically, the conventional wisdom is that in a well-engineered network the RNC network is unlikely to be the bottleneck relative to the air interface (wireless hop) [22].

For the purpose of deployablity, we do not assume that we have vantage points within the cellular network or along the WAN path for additional measurements, nor do we assume access to a cohort of cooperating clients. Hence, we focus on designing an end-to-end measurement technique between a server on the Internet and an individual cellular-connected mobile device. Moreover, since cellular connections are metered, a key constraint is to ensure that the data usage incurred for our measurements is quite low and is unlikely to exacerbate congestion.

Our goal is to have a technique that works across different platforms and does not require any new OS or network capabilities, and can be easily deployed in existing platforms as a user-level application. This is different from prior techniques such as LoadSense [3], which relies on passive low-

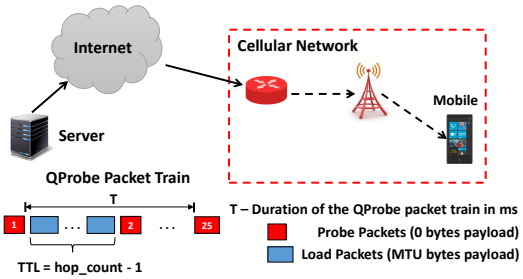

Figure 2: QProbe packet train with tiny probe packets and large load packets. TTL for load packets expires 1-hop before cellular network and get dropped en route.

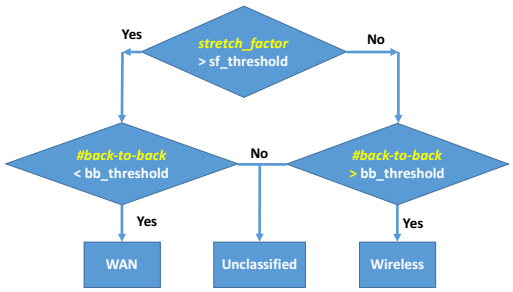

Figure 3: QProbe Algorithm

level measurements from the cellular stack.

3.2 Detection Methodology

At a high level, QProbe technique consists of actively sending probe traffic downstream from an Internet server to a mobile device, and analyzing the probes received at the client-side to diagnose the problem. We send a train of equally spaced tiny probe packets to the client and observe the arrival times. The small-sized probes ensure that the probing traffic is light.

When these packets arrive at the basestation, if the wireless link is not congested, the basestation would not be backlogged and so the packets are likely to be delivered immediately. In this case, we expect to see the inter-packet spacing at the receiver to be very similar to that at the transmitter. However, with an increase in wireless cross-traffic, the basestation would have to service several other clients, and the probe packets are likely to get queued up at the basestation. Two successive probes would get queued up at the basestation if the inter-packet spacing at the sender is smaller than the times when the client gets scheduled by the PF-scheduler. When the PF-scheduler chooses to service the client to which the probe traffic is destined, multiple probe packets can be delivered depending on the quantum of service allocated to the client.

Moreover, since the packets are small-sized, the chances of multiple probe packets being delivered back-to-back are very high. Thus, to detect wireless congestion, we look for occurrences of increase in inter-packet spacing followed by one or more back-to-back packets[1]. As observed experimentally in the previous section, the higher the congestion, the larger the spacing between two consecutive scheduling opportunities. Thus, we expect to see more back-to-back packets delivered during each scheduling opportunity.

The idea of using a number of back-to-back probes as a measure of wireless congestions relies on the fact that probes arrive spaced apart in time at the basestation. However, cross-traffic on the WAN path can also cause probe packets to get queued up at a WAN link, and arrive at the basestation without any gaps between them. In such a case, differentiating the WAN effects from wireless congestion is difficult.

[1]When two consecutive probes arrive within a single transmission time interval (2 ms), we count them as back-to-back.

To isolate the effects of WAN cross-traffic, we introduce *load packets*, one or more large MTU sized packets in-between two successive probes. The objective of the load packets is to ensure that the probe packets arrive spaced apart in time at the basestation. But, load packets introduce two new problems. Since they are large, the data consumption for measurement traffic increases. More importantly, at high wireless congestion, as the per-client allocation during each scheduling opportunity reduces, multiple probe packets may not get scheduled at the same time even if they are queued up at the basestation. To address these problems, we employ a commonly used TTL-based approach to drop the load packets at an IP hop close to the cellular network before they reach the basestation. We set the TTL of the load packets to $hop\_count - 1$, where hop_count is the number of hops in the path between the server and the client.

Load packets provide two benefits. Not only do they ensure that probe packets arrive spaced apart in time at the receiver, they also help in detecting WAN link bottlenecks. In cases where the WAN link is the bottleneck, the load packets can introduce additional delay due to the packet transmission time of MTU-sized packets over a wired bottleneck link. As a result, the packet spacing between successive probe packets can increase. We measure this effect using a metric called *stretch-factor*, which is computed as the ratio of time duration of the packet train at the receiver to that at the sender. The duration of the train refers to the time between the last and the first probe packets.

Ideally, when there are no bottlenecks anywhere, the packet spacings are similar, and the *stretch-factor* should be close to 1. When there are bottlenecks in the WAN path, additional spacing is introduced due to the transmission of the load packets, and the total duration of the train gets stretched at the receiver, resulting in a *stretch-factor* value greater than 1. With wireless bottlenecks, we still expect low *stretch-factor* (close to 1) since the PF-scheduler can deliver multiple probe packets back-to-back after each occurrence of a large gap due to cross-traffic at the base station.

The overall construction of the QProbe packet train is shown in Figure 2. We send a sequence of 25 probe packets spaced equally apart. Multiple load packets of MTU size with TTL set to $hop\_count - 1$ are sent in between two consecutive probe packets. The spacing between two load packets as well as the spacing between a probe packet and a load packet is set to 1 ms. The user-level QProbe client appli-

cation records the timestamps of the received packets and runs our detection algorithm to determine the bottleneck location. Since we do not know the wireless conditions, we run QProbe train for different values of spacings (4ms to 8ms) between the probe packets to get a better estimate of the condition. Therefore, the total data usage for running 5 QProbe trains is only 3.5KB, and completes in just 720 ms.

Our detection algorithm uses two features to determine whether it is a wireless problem or a WAN path problem. When the number of back-to-back packets observed is high and the *stretch-factor* metric is low, we classify it as a wireless problem, and whenever the number of back-to-back packets is low and the *stretch-factor* is high, we classify it as a WAN bottleneck. Otherwise, we let the problem remain unclassified.

For each QProbe run for a certain probe packet spacing, we record the packet arrival timestamps, and run the detection algorithm. The detection algorithm for QProbe is described in a flow-chart shown in Figure 3. There are two thresholds – sf_threshold and #bb_threshold that are used to determine the different bottleneck scenarios.

Computing QProbe Thresholds: Using controlled experiments (see Section 4.2), we evaluated QProbe by varying packet spacings from 4 ms to 8 ms in increments of 1 ms for 489 problem cases, out of which 233 were bottlenecked by the wireless link and the remaining 256 cases experienced WAN bottlenecks. For each of these runs, the ground truth is known – whether the problem is wired or wireless. We trained a 10-fold cross-validation decision tree to predict the bottleneck using *stretch-factor* and the number of back-to-back packets as the two features. We use the thresholds obtained from the tree that is generated from this training phase for classification of QProbe runs done on 3G.

Since LTE throughput ranges are different from 3G, the thresholds for LTE are trained separately. However, we did not have a sufficient number of LTE connections to perform controlled runs. Therefore, we used a subset of the LTE runs that we obtained from our large-scale measurement study (Section 4.3) to build decision tree models for LTE, and derived the thresholds for the two metrics.

Reasons for Unclassified: There can be instances of wireless bottleneck cases that have high *stretch-factor* along with high back-to-back packets. These occur mostly due to transient congestion occurring in both wired and wireless paths. In addition, some wired bottleneck cases can have a low *stretch-factor* and low back-to-back packets. We believe, this can occur due to congestion in the wired backhaul portion of the cellular network. Since, load packets get dropped at the IP cellular gateway, any congestion in the wired path beyond this point are unlikely to impact the probe packets and thus does not cause the two metrics to increase significantly. Both the above two categories are hard to identify due to lack of ground truth information. In our algorithm, we treat them as unclassified runs.

4. EVALUATION

We validate the QProbe technique using LTE simulations in NS3, and controlled experiments in two 3G networks in India. We then go on to deploy QProbe as an iPhone app and collect data from the wild for a large number of cellular operators for both 3G and LTE networks.

4.1 Simulations

Using NS3, we created a real-world topologies consisting of a multi-hop wired WAN path that connected to an LTE basestation having a PF-scheduler. The WAN path contained 17 hops, which is the mean number of hops in the data we gathered from the wild (more details in Section 4.3). The link speeds of the WAN path were set to 1 Gbps. We connected a server on the wired endpoint to generate probing traffic destined to a cellular client at the other endpoint. We then created different wireless congestion levels by increasing the number of background bulk TCP downloaders in the cell, varying the load level from low (6 downloaders) to medium (9 downloaders) to high (13 downloaders). The number of downloaders for each congestion level was decided such that the TCP throughput observed at the QProbe client resembled the 75^{th}, 50^{th}, and 25^{th} percentiles of the TCP throughputs we observed in the LTE measurements of our dataset (Section 4.3). As part of QProbe, we sent a 100 ms probe train consisting of 25 packets with 4 ms inter-packet spacing from the server.

Figure 4(a) shows the number of back-to-back packets received at the client side, and Figure 4(b) shows the corresponding *stretch-factor* for the same conditions. At high loads, almost 80% of the probe traffic arrive back-to-back, thus matching our hypothesis. In addition, *stretch-factor* remains constant for the most part, with only a marginal increase of 10% at high load. Thus, both metrics behave as expected with wireless congestion.

We analyze the benefit of using load packets in between probe packets in distinguishing WAN and wireless bottlenecks. An intermediate WAN link can become a bottleneck for two reasons: (a) due to low capacity, and (b) due to low available bandwidth owing to cross-traffic. Our objective is to detect WAN bottlenecks for both these cases.

We varied the capacity of an intermediate WAN link from 1 Mbps to 5 Mbps. As seen in Figure 4(c), the *stretch-factor* does not change when there are no load packets in the probe train, but with the introduction of load packets, the stretch factor increased by a factor of 9 when the bottleneck bandwidth is as low as 1 Mbps. Similarly, in the cross-traffic scenario, with the increase of the volume of cross-traffic, Figure 4(d) shows that the *stretch-factor* shows an increase of more than 25%.

4.2 Controlled Experiments

To evaluate QProbe on real networks, we conducted controlled experiments on two 3G operators – BSNL and Airtel in Bengaluru, India, with several runs corresponding to the wireless and WAN bottleneck scenarios. Using five co-located smartphones, we generated heavy background wireless traffic in the cell via simultaneous TCP bulk downloads. Next, we connected a laptop running the QProbe client to the same basestation using a 3G USB dongle, and conducted

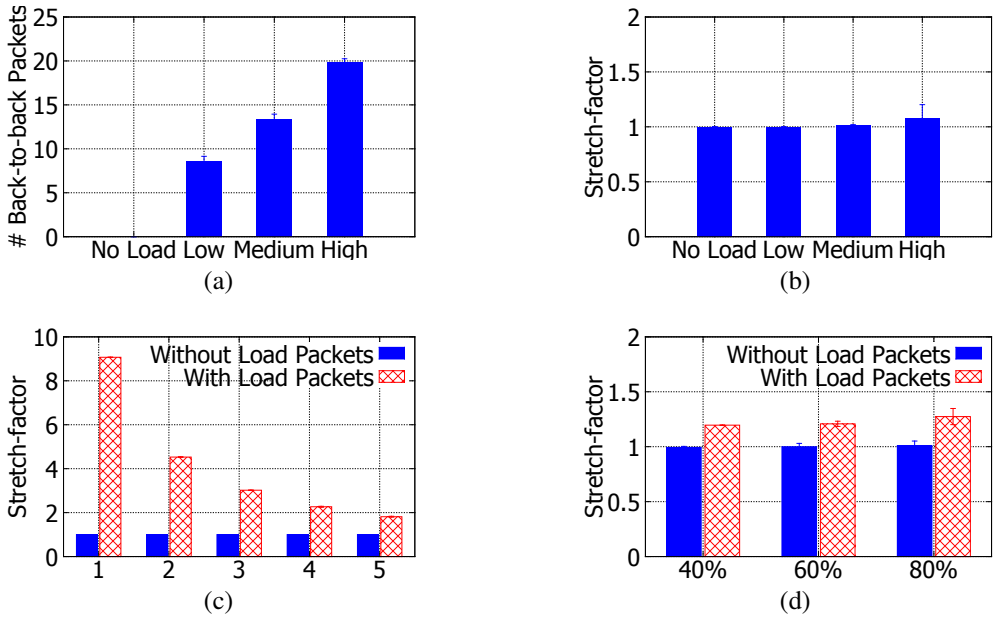

Figure 4: (a) Number of back-to-backs, and (b) *stretch-factor* with varying wireless congestion levels. *stretch-factor* for WAN link bottlenecks due to (c) varying link bandwidth (in Mbps) and (d) varying cross-traffic volumes.

233 QProbe runs from a well-provisioned server hosted on the Azure datacenter in Singapore. Note that these runs are bottlenecked by the wireless link due to the presence of background downloaders. To gather runs for WAN bottlenecks, we also deployed the QProbe server on 34 Planetlab servers that had bottlenecked WAN paths[2] and conducted 256 QProbe runs from them. To minimize interference from other traffic in the same cell, we ran these experiments late at night. To ensure the load packets are not accounted for byte usage, we had to set the TTL to $hop - count - 2$.

Figure 5 depicts the *stretch-factor* and the number of back-to-back packets of all the runs we conducted for a 4ms inter-packet spacing for the probe packets. This plot verifies our simulation results, in that we see more back-to-back packets and a small *stretch-factor* in the presence of a wireless bottleneck, whereas a wired bottleneck result in fewer back-to-back packets and a higher *stretch-factor*.

Using the thresholds obtained from the training phase (Section 3.2), QProbe algorithm classified 94.7% of the 489 runs, while 5.3% of the runs did not satisfy the two threshold conditions, and thus remained unclassified for reasons described in Section 3. For the runs that were classified, the accuracy of detecting both wired and wireless bottlenecks is more than 97.4%, thus showing that with a simple, easy to measure client-side algorithm, we can accurately detect the bottleneck location.

For validation, we also ran QProbe without background traffic at night time and collected 833 runs for 3G and 989 runs for LTE. QProbe indeed classified only 8.2% and 1.9% as wireless bottlenecks for 3G and LTE, respectively, since these periods are lightly loaded.

[2]We verified this by running TCP measurements with these servers using a high-bandwidth wired link from the same service provider as the cellular ISP.

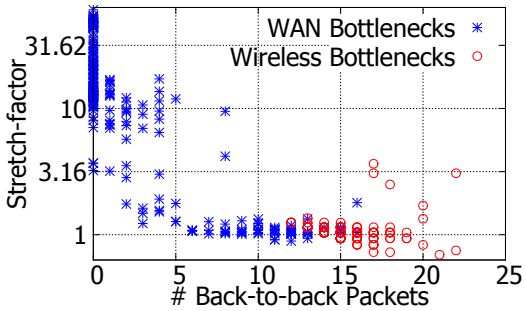

Figure 5: # Back-to-back packets and *stretch-factor* for 489 QProbe runs. (Y-axis is in log scale.) WAN bottlenecks result in low # back-to-back packets and high *stretch-factor* while wireless bottlenecks show the opposite behavior.

4.3 Large-scale Measurement Study

We developed QProbe as an iOS application on both iPads and iPhones, and deployed it in the Apple App Store [16]. To detect bottlenecks in different end-to-end paths, we used 15 well-provisioned Microsoft Azure servers that were deployed in different datacenters, located in the US, South America, Europe, Singapore, China, Japan, and Australia. We also deployed the QProbe server on 51 geographically spread PlanetLab servers.

We collected data from this deployment via users of the QProbe iOS application. We advertised the app through social platforms as well as Amazon MTurk to get more par-

Table 1: Summary of the dataset.

#Users	642
Data collection period	2 months
#QProbe Runs	8116
#Countries	33
#Cellular Providers	51

Table 2: QProbe runs for different radio technologies

Technology	Runs	Wireless Bottlenecks	WAN Bottlenecks
3G	2573	215 (8.4%)	97 (3.8%)
LTE	5480	441 (8.1%)	837 (15.3%)

Table 3: Confusion matrix for 3G

		QProbe Classification	
Ground Truth		Wireless	WAN
Wireless	187	161 (86.1%)	26 (13.9%)
WAN	76	13 (17.1%)	63 (82.9%)

Table 4: Confusion matrix for LTE

		QProbe Classification	
Ground Truth		Wireless	WAN
Wireless	330	307 (93%)	23 (7%)
WAN	708	116 (16.4%)	592 (83.6%)

ticipation [3]. Table 1 summarizes our dataset. We have made our dataset publicly available at [15].

The 8116 runs were spread across different cellular radio access technologies as shown in Table 2. We only analyzed data from the dominant radio technologies present in this study: 3G (WCDMA, HSDPA, and EVDO RevA) and LTE.

When users run the QProbe app, the app connects to the two closest Azure servers, referred to as reference servers, based on TCP RTTs. Then, the app conducts bulk TCP download measurements with these two reference servers sequentially. Next, the app runs QProbe experiments from 6 randomly chosen servers sequentially. Thus, each run of the app provides 6 QProbe measurements. For each of these 6 servers, the application first does TCP throughput measurements and then receives 5 QProbe packet trains, having probe packet spacings of 4 ms to 8 ms in increments of 1 ms. It logs the arrival timestamps of the probe packets and uploads these timestamps, along with throughput measurements and carrier information like the MNC, MCC codes, and the operator's name to a central server. We analyze this data offline to evaluate QProbe's bottleneck detection accuracy.

Obtaining the Groundtruth: We use multiple throughput measurements conducted during each run of the app to estimate ground truth. The basic idea is that whenever low throughput is consistently observed with both reference servers and the server the QProbe train is sent from, we blame it on the last-mile wireless link. Otherwise, we blame the specific instance of low throughput on the WAN path from that specific server. To determine what constitutes low throughput, we looked at all throughput measurements obtained for 3G and LTE networks separately, and chose the 25th percentile as the threshold (732 kbps and 3149 kbps for 3G and LTE, respectively).

Bottleneck Detection Accuracy: Using the threshold parameters obtained from training, we classified the bottlenecked runs in the measurement study. QProbe classified 84.3% and 81.2% of 3G and LTE runs, respectively. For those that were classified as either wireless or WAN path, the overall accuracy of bottleneck detection is over 85% for both 3G and LTE. Table 3 and Table 4 provide the confusion matrix for 3G and LTE, respectively.

[3]We obtained Microsoft IRB approval for this study.

The Need for Two Metrics: Note that if we were to use the number of back-to-back packets or the *stretch-factor* in isolation to classify the runs, all of them can be classified. However, in doing so, the classification accuracy drops significantly. For example, if QProbe used only the number of back-to-backs or the *stretch-factor* for classification, the accuracy reduces by 14.4% and 15.5%, respectively. For 3G, the two corresponding accuracy reductions are 14.6% and 21.7%. Therefore, though the two metric based classification does not classify a fraction of the runs, it achieves a far higher classification accuracy than using a single metric.

5. RELATED WORK

There has been much work on determining the location of the bottleneck link in both wireless and wired settings due to the numerous benefits such techniques provide. For example, a technique like DiversiFi [10] can be used to improve the reliability of real-time streams (e.g., Skype calls) if users can determine that their WiFi link is poor. In the WiFi context, there has been recent work [21, 20] on correlating TCP performance with wireless metrics to determine where the bottelneck lies in home networks. In cellular context, passive analysis of TCP flows inside the operator's network [18] have been used for identifying bottlenecks, but such vantage points for measurements are not available to end-hosts. Hu *et al.* present a technique in [7] that is capable of locating bottlenecks in the Internet. Other work has focused on diagnosing problems on WiFi, e.g., [1, 17, 9]. However, cellular networks tend to be closed and do not lend themselves to such analyses.

There is a large body of work on probing network paths to determine the capacity (the speed of the slowest link) and available bandwidth (the headroom on the tightest link).

The literature on capacity estimation often uses the *packet-pair technique* [11], where a server sends two back-to-back packets and the receiver estimates the capacity based on the time gap of the packet arrivals. As this technique is susceptible to cross-traffic interference, other work has employed packet trains of various sizes [2], or filtering to discard samples that do not relate to the bottleneck link capacity [13]. Another widely used technique is based on the relationship between the packet size and the delay to estimate the capacity [5, 14]. A shortcoming of these tools is their dependence on ICMP messages that are often blocked.

The literature on measuring available bandwidth can be mainly divided in to two groups: packet rate method (PRM) [8, 6] and packet gap method (PGM) [19]. The PRM method works by sending trains of probe packets at different rates, and checking at what point the receiving rate stops keeping up. A significant downside, however, is that these tools tend to impose a heavy load on the network. In contrast, PGM tools estimate the change in spacing between pairs of equal-sized probe packets as seen at the receiver, which is an indication of the volume of cross-traffic. While it is lighter weight than PRM, PGM's estimation depends critically on FIFO queuing being employed at the bottleneck.

Prior work has shown that above techniques do not provide accurate results in cellular networks [23]. Therefore, our work on `QProbe` is distinguished from the above body of work in terms of its focus on (a) locating the bottleneck link rather than estimating its capacity, (b) cellular-terminated paths, and (c) a deployable tool that can run on off-the-shelf client devices (e.g., iPhone), which do not provide access to any low-level network information.

6. CONCLUSION

Cellular-connected users often blame poor end-to-end network path performances on the last-mile wireless connection. However, as shown in our measurement study, the WAN path can be the bottleneck in many cases. `QProbe` locates the bottleneck location by leveraging the unique characteristics of the PF-schedulers used in cellular basestations. While being a lightweight probing technique requiring only $\sim$ 700 ms to run and less than 4KB data usage, `QProbe` locates the bottleneck in real-world 3G and LTE networks with more than 85% accuracy.

Acknowledgments

We thank the anonymous ACM CoNEXT 2015 reviewers for their constructive feedback. Author Baranasuriya was an intern at Microsoft Research India during part of this work. He and author Gilbert were supported in part by A*STAR, Singapore, under SERC Grant 1224104049.

7. REFERENCES

[1] A. Adya, P. Bahl, R. Chandra, and L. Qiu. Architecture and Techniques for Diagnosing Faults in IEEE 802.11 Infrastructure Networks. In *MobiCom*, 2004.

[2] R. L. Carter and M. E. Crovella. Measuring Bottleneck Link Speed in Packet-switched Networks. *Perform. Eval.*, 1996.

[3] A. Chakraborty, V. Navda, V. N. Padmanabhan, and R. Ramjee. Coordinating Cellular Background Transfers Using Loadsense. In *MobiCom*, 2013.

[4] S. Deng, R. Netravali, A. Sivaraman, and H. Balakrishnan. WiFi, LTE, or Both? Measuring Multi-Homed Wireless Internet Performance. In *IMC*, 2014.

[5] A. B. Downey. Using Pathchar to Estimate Internet Link Characteristics. In *SIGMETRICS*, 1999.

[6] A. B. Downey. Using pathchar to Estimate Link Characteristics. In *SIGCOMM*, 1999.

[7] N. Hu, L. E. Li, Z. M. Mao, P. Steenkiste, and J. Wang. Locating Internet Bottlenecks: Algorithms, Measurements, and Implications. In *SIGCOMM*, 2004.

[8] M. Jain and C. Dovrolis. End-to-end available bandwidth: measurement methodology, dynamics, and relation with tcp throughput. In *SIGCOMM*, 2002.

[9] P. Kanuparthy, C. Dovrolis, K. Papagiannaki, S. Seshan, and P. Steenkiste. Can User-Level probing Detect and Diagnose Common Home-WLAN Pathologies. *ACM SIGCOMM Computer Communication Review*, 2012.

[10] R. Kateja, N. Baranasuriya, V. Navda, and V. N. Padmanabhan. DiversiFi: Robust Multi-Link Interactive Streaming. In *CoNEXT*, 2015.

[11] S. Keshav. A Control-theoretic Approach to Flow Control. In *SIGCOMM*, 1991.

[12] H. Kushner and P. Whiting. Convergence of Proportional-fair Sharing Algorithms Under General Conditions. *Wireless Communications, IEEE Transactions on*, July 2004.

[13] K. Lai and M. Baker. Measuring Link Bandwidths Using a Deterministic Model of Packet Delay. In *SIGCOMM*, 2000.

[14] pchar: A Tool for Measuring Internet Path Charateristics (by B. Mah). http://www.kitchenlab.org/www/bmah/Software/pchar/.

[15] QProbe Project Page. http://www.comp.nus.edu.sg/ nimantha/qprobe.html.

[16] QProbe-Cellular Link Diagnostic Tool. https://itunes.apple.com/sg/app/qprobe-cellular-link-diagnostic/id988472779.

[17] S. Rayanchu, A. Mishra, D. Agrawal, S. Saha, and S. Banerjee. Diagnosing Wireless Packet Losses in 802.11: Separating Collision from Weak Signal. In *INFOCOM*, 2008.

[18] M. Schiavone, P. Romirer-Maierhofer, F. Ricciato, and A. Baiocchi. Towards Bottleneck Identification in Cellular Networks via Passive TCP Monitoring. In *ADHOC-NOW*, 2014.

[19] J. Strauss, D. Katabi, and F. Kaashoek. A Measurement Study of Available Bandwidth Estimation Tools. In *IMC*, 2003.

[20] S. Sundaresan, N. Feamster, and R. Teixeira. Measuring the Performance of User Traffic in Home Wireless Networks. In *PAM*, 2015.

[21] S. Sundaresan, Y. Grunenberger, N. Feamster, D. Papagiannaki, D. Levin, and R. Teixeira. WTF? Locating Performance Problems in Home Networks. *Tech Report GT-CS-13-03*, 2013.

[22] D. Wisely. *IP for 4G*. John Wiley & Sons, 2009.

[23] Y. Xu, Z. Wang, W. K. Leong, and B. Leong. An End-to-End Measurement Study of Modern Cellular Data Networks. In *PAM*, 2014.