Passive Mobile Bandwidth Classification Using Short Lived TCP Connections

Foivos Michelinakis[12], Gunnar Kreitz[3], Riccardo Petrocco[3], Boxun Zhang[3], Joerg Widmer[1]
[1]IMDEA Networks Institute, [2]Universidad Carlos III de Madrid, [3]Spotify
{foivos.michelinakis, joerg.widmer}@imdea.org,
{gkreitz, r.petrocco, boxun}@spotify.com

***Abstract*—thereConsumption of multimedia content is moving from a residential environment to mobile phones. Optimizing Quality of Experience—smooth, quick, and high quality playback—is more difficult in this setting, due to the highly dynamic nature of wireless links. A key requirement for achieving this goal is estimating the available bandwidth of mobile devices. Ideally, this should be done quickly and with low overhead. One challenge is that the majority of connections on mobiles are short-lived TCP connections, where a significant portion of data exchange is within the slow start phase. In this paper, we propose a novel method that passively estimates the currently available bandwidth by monitoring the minimal traffic generated by such connections. To the best of our knowledge, no other solution can operate with such constrained input. Our estimation method is able to achieve good precision despite artifacts introduced by the slow start behavior of TCP, mobile scheduler and phone hardware. We evaluate our solution against traces collected in 4 European countries. Furthermore, the small footprint of our algorithm allows its deployment on resource limited devices.**

Keywords*—*Mobile bandwidth measurement, LTE

I. Introduction

In recent years, the availability of fast cellular networks has led to increasing usage of mobile devices. Thus, the distribution of audio and video content is swiftly moving from the desktop environment to the mobile one. This shift affects streaming applications like Spotify [13], that have seen an increasing mobile consumption over the last years and continue to do so. In the competitive market of streaming services, Quality of Experience (QoE) is important. The metrics comprising QoE include fast start-up time, low playback latency, stutter-free media delivery, and high bit-rates.

Given the high variation of the available radio resources and capabilities of mobile devices, streams are often available in different bit-rates. To achieve high QoE, it is imperative to select the highest possible bit-rate, as constrained by available bandwidth and device hardware. Ideally, it is desirable to avoid switching bit-rates during playback, but due to the difficulty of predicting bandwidth, it is currently a common practice to begin playback on a low-quality bit-rate, and then increase it if possible. In order to improve on the current practice and select an appropriate bit-rate also for the initial part of stream, it is required to have a bandwidth estimation algorithm that operates on very small amounts of traffic.

Another key use case for a rapid bandwidth estimator is to make buffering decisions. Streaming media players need to buffer content before commencing playback, and thus need to decide on the size of said buffer. Such decisions require an estimate of the bandwidth available in the near future. This is particularly important for applications such as Spotify, which do not perform bit-rate switching, but still require low playback latency. Furthermore, when streaming audio tracks, the small amount of data transferred makes it difficult to apply traditional bandwidth estimators. Spotify reported in a previous study [13] a median playback latency of 265 ms with less than 1% of streams suffering one or more stutter events. However, at that point, most users were streaming on desktops rather than mobile devices, and the data reported did not include phones.

The current state of the art solutions for bandwidth estimation require the transmission of link saturation traffic [19], [8], [22], a practice that is problematic considering the scarce resources of mobile devices. The focus of this study is the estimation of available bandwidth of mobile phones through passive traffic monitoring. We specifically focus on traffic generated during the initial few hundred milliseconds of the TCP "slow start" phase. At least 95% of the streams a mobile phone generates are TCP and the majority of them are so short lived that they do not exit the slow start phase of TCP [9]. In this paper, we present a method that provides good bandwidth estimation in these challenging conditions. Further, it is robust against measurement artifacts introduced by hardware-limited mobile devices, and the scheduling process of the base station. The run time complexity of our algorithm is $O(n)$ and requires only the first packets of a TCP stream.

The output of our algorithm can also be used as input to bandwidth optimization algorithms [16], [1], [4], to achieve more efficient usage of the mobile resources. Such algorithms are usually based on a resource prediction model that makes heavy use of time series information. We evaluate our algorithm by means of real-word experimentation, through LTE traces collected in 4 European countries (Germany [12], Sweden, Greece and Spain), using a variety of Android devices.

The remainder of this paper is structured as follows: related work is introduced in Section II and some essential theoretical background fundamentals are analyzed in Section III. Our algorithm and a comparison with similar tools are presented in Sections IV and V and their discussion is in Section VI. Finally, Section VII summarizes our conclusions.

II. Related work

Nowadays, the most popular solution for mobile bandwidth estimation is Ookla's mobile application *Speedtest* [19], which makes use of active measurement techniques. To estimate the bandwidth available to a mobile device, Speedtest tries to saturate the device's downlink by downloading a large file through two parallel long-lived TCP connections in a test that lasts for about 10 seconds. Speedtest can connect to many well-connected measurement servers deployed by Ookla, so the measured bandwidth is less likely to be affected by cross traffic. To the best of our knowledge, there is no public documentation detailing the algorithm used by Speedtest on mobile. Our understanding is that every second, a certain amount of traffic samples are generated. These samples are aggregated into 20 bins, which are then filtered to remove measurement artifacts. The final estimation is calculated by averaging over the bandwidth values of the remaining bins. A similar approach is proposed by [8], where 3 parallel TCP connections are established with the three closest servers to reduce the impact of TCP's receive window, overloaded servers, and packet losses. The traffic samples are segmented into equally sized bins, and samples collected during the slow-start phase are discarded. Then, the median of the bandwidth estimated in the remaining bins is taken as the final estimation. In [22], authors calculate the end-to-end throughput availability by sending high rate UDP traffic, while taking into consideration scheduling effects of mobile networks. These tools are unsuitable for frequent usage on mobile devices as they rely on transfer of large amounts of data.

"Packet dispersion" is a lightweight active measurement technique [14], [6]. Packet pairs or packet trains are transmitted from a server to a target device, which timestamps their arrivals. It is meant to measure the asymptotic capacity of the path's bottleneck link by analyzing the time dispersion of packet arrivals. As we showed in [17], applying this technique in a mobile network is problematic because the scheduler of the base station either shrinks or enlarges the dispersion greatly. A packet pair dispersion technique that is able to operate in mobile networks is presented in [11]. This approach though is very sensitive to topology parameters, like the number of nodes in the path and the link utilization. The number of packet pairs required to generate a valid estimation, rapidly increases, when these parameters increase. Furthermore, it ignores the effects of the mobile scheduler.

On the other hand, several approaches that estimate available bandwidth by monitoring mobile communications but without generating any traffic have been proposed. The method proposed in [7] first sniffs all the traffic going through a certain vantage point inside an operator's network. Then it identifies the traffic flows that belong to applications that are not rate-limited at the server side and are guaranteed to use all the resources that the base station allocates to a user. A similar approach is proposed in [20] and [9] for UMTS and LTE networks, respectively. Even though all these techniques are passive, they are implemented inside the operator's network. Therefore, they are not applicable at the application level and require the cooperation of the operator.

We conclude that, even though available bandwidth estimation is a well studied topic, none of the proposed solutions are able to provide trustworthy results on the device side by just relying on the minimal traffic generated during the slow start phase of TCP.

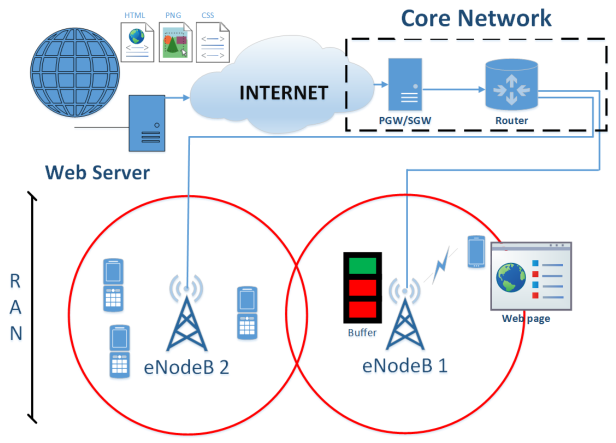


Fig. 1: Simplified LTE topology and network components that form the path between a server and a mobile device.

III. Theoretical Background

This section provides an introduction to the technologies and characteristics of the current generation of mobile networks. We focus on LTE, due to its increasing popularity.

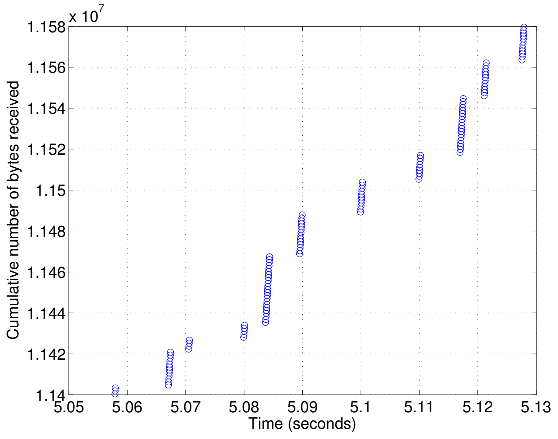
A. *LTE fundamentals*

When a mobile user surfs the web through an User Equipment (UE), a TCP connection is established with a remote server. As shown in Figure 1, the packets associated with this TCP connection travel through the Internet, enter the core network of the mobile operator through the Packet Data Network Gateway (PGW), and are then routed to the serving base station (eNodeB) that the UE is connected to. At the eNodeB, the packets are stored in a buffer dedicated to the target device. The allocation of resources to the connected UEs is determined by a scheduling mechanism, which considers the recent resource allocation history, the channel quality of devices that have pending traffic, and some fairness conditions. This mechanism tries to find a balance between sending as much data as possible and fulfilling the needs of all connected UEs.

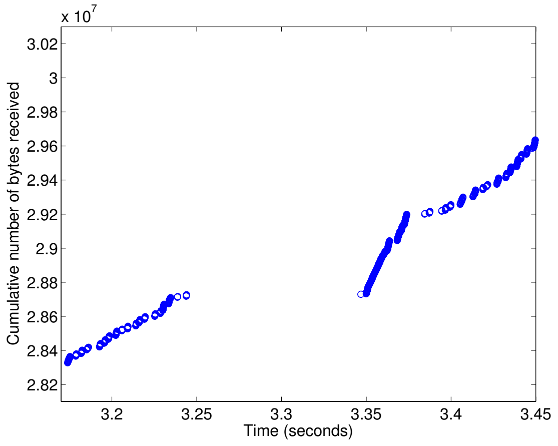
The scheduling decision is taken periodically, once every Transmission Time Interval (TTI). The TTI of the downlink for the Frequency-Division Duplexing (FDD) version of LTE, which is used by the majority of operators, is fixed to 1 ms [10]. For the Time-Division Duplexing (TDD) version, used mostly by operators in China [5], the TTI is in the range of a few ms, depending on operator configuration. When a UE is scheduled, the packets present in the buffer are grouped into a Transport Block (TB), which is then sent to the UE. In case of a very bad signal and/or a small amount of allocated resources, a segment of a packet can be encapsulated in a TB instead.

B. *Measurement accuracy*

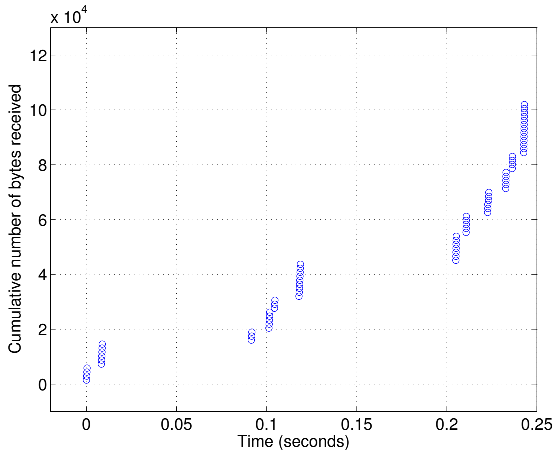
Our tool is based exclusively on phone side measurements, and this imposes some limitations. Ideally, we would like to measure the exact size and timing of every TB that is being allocated to the device. However, this would require information that is only available at the eNodeB, or at the Network Interface Card (NIC) of the mobile device. Extracting such information from the NIC is impossible without specialized



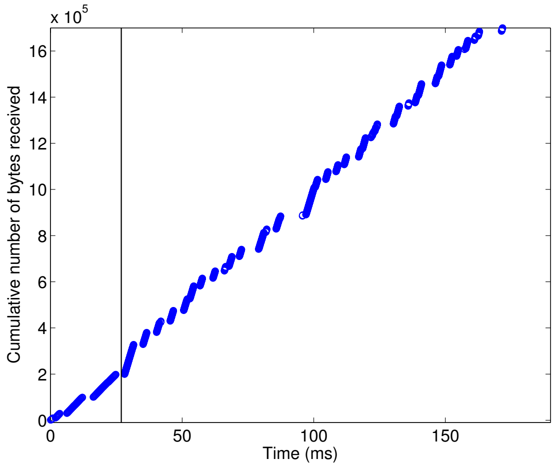
(a) FDD LTE device receiving saturation traffic.



(b) Artifact caused by a weak and/or busy CPU.



(c) During TCP slow start the server sends packets in distinct groups. Three such groups are visible in the figure.



(d) Arrival of high speed UDP constant bit-rate traffic. Packets on the left side of the vertical line are arriving slower than those on the right side.

Fig. 2: IP packet arrivals to an LTE device for different scenarios. The first data packet arrives at time 0.

drivers that vendors are very hesitant to release for public usage. Instead, we use rooted Android phones and the traffic sniffing program TCPdump [21] to record the time and the size of every IP packet reported by the kernel. Unfortunately, a noticeable delay exists between the arrival of an IP packet at the NIC (as part of a TB) and its appearance in the kernel. As we discuss below, such delay can potentially impact the accuracy of our method.

An attempt to measure a similar delay is presented in [15]. The authors try to measure the delay between the arrival of an IP packet at the WiFi interface, and its registration in the kernel. They believe that this delay should be comparable with the delay between the LTE NIC and the kernel. The physical link or the driver reacts faster to TCP data packets compared to ICMP and TCP SYN/RST. Thus, the data packets that we target have the lowest possible delay. The delay is dependent on the network card and varies from insignificant (hundreds of μs), to being variable in the range of a few ms. According to [22], both delays are related to the polling frequency of the NIC from the OS. TCPdump registers packets as soon as they arrive in the kernel, but in case of high network load, the kernel might choose to delay polling in order to reduce packet processing overhead.

We have investigated the impact of polling in a variety of phones through a small scale experiment. When a cellular network is used, all the phones we tested report packets in groups, similar to the ones shown in Figure 2a. The exact size and spacing of the groups varies depending on how powerful the phone is, but the pattern is always similar. Thus, the pattern is determined by the grouped delivery of packets in the physical layer, but the timestamping accuracy of each packet is related to the phone hardware. On the other hand, under WiFi (802.11g without packet coalescing), different phones exhibit very different behavior. In this scenario, some phones report the packets in the same grouped fashion, whereas others report continuous delivery of packets. A sniffer we used to monitor the WiFi traffic while conducting the experiment always detected a continuous delivery of packets “in the air”. Based on these observations, we conclude that the pattern of packet arrival on WiFi seems to be greatly dependent on the phone specifications. Exhaustively studying this effect and adapting our algorithm to it is beyond the scope of this paper.

C. Measurement artifacts

Figure 2 presents several characteristic packet arrival patterns captured by TCPdump running on a FDD LTE UE.

a) Scheduler: When the UE is scheduled by the eNodeB, it receives packets during one or more TTIs. When it is not, no packets are registered. In Figure 2a, the scheduler effect can be easily observed.

b) CPU: When the CPU is busy, it may significantly delay the polling of the NIC, as shown in Figure 2b. This figure presents the steady state phase of a TCP connection, where packets were continuously arriving to the NIC. The kernel did not poll the NIC for about 100 ms, which caused a "gap" in the registering arrivals of packets.

c) TCP slow start: The perhaps most well known artifact which our algorithm must handle is the grouping of packets during the slow start phase of a TCP connection. After the handshake is finished, the server sends data packets back-to-back, until the number of unacknowledged packets reaches the congestion window value, which typically starts small. These packets, unless there is significant cross traffic in the path between the server and the UE, arrive as a group in the eNodeB, which buffers them. Depending on the group size, the channel quality of the target UE and the cell congestion, the eNodeB might not be able to transmit all packets during one TTI, but might require several TTIs, spread across a few tens of ms. Upon reception, the UE transmits the related ACKs. In both FDD and TDD, the transmission opportunities for the UE to send data are at least 8 ms apart (in TDD even more). Thus, the TCP ACK packets are sent in groups. Upon arrival at the server, the ACK group triggers the transmission of a larger group of data packets to the UE, since the congestion window now has a higher value. The delay between the arrival of such data packet groups at the antenna is usually higher than their transmission time to the UE, thus the data packet groups can be identified at the client side. The first three such groups of a TCP flow can be seen in Figure 2c. Notice that each subsequent group has increased size, since the congestion window takes progressively larger values. Eventually, the congestion window reaches the size of the bandwidth delay product and thus TCP is able to send continuously. As a consequence, the packet delivery at the phone side then becomes continuous. The specific flavor of TCP is not important, since all of them use initial congestion window values that create this effect.

d) Slower arrival of the first packets: We were able to identify a significant difference in the downlink speed during the first few hundred packets, compared to the speed achieved later on the same flow, in cases where the UE may achieve a very high speed. We have observed this effect in all our traces, gathered in four European countries with a variety of phones. The number of incoming packets in the middle of the trace is higher compared to the beginning, even after taking into consideration the reduced slow start TCP speed. We further investigated this in a Spanish network, by sending constant bit-rate UDP traffic to a UE. We did not observe any difference in the downlink speed when the server speed was less than 25 Mbps. When the server was transmitting at rates higher than 25 Mbps though, the first packets (ranging from the first 150 to the first 300 packets) were received at about 25% to 50% lower speed compared to the rest. For example, in Figure 2d the first 190 packets (left of the vertical line) were received at a 25% lower speed compared to the remaining packets (shown at the right of the vertical line). If the transmission is paused for a few tens of ms, we observe the same pattern when the transmission starts again. A similar effect was observed by an independent group, which did measurements in the same German network we used to collect our traces [3]. According to this study, when the speed is higher than 20Mbps, the first packets of a flow are delivered with considerably higher delay than the remaining ones. Since we do not have physical layer or mobile network specific information, we can not know the exact cause of this behavior. We suspect that it is an operator configuration. Finally, we have indications that this phenomenon is even more prominent in 3G networks, but 3G is beyond the scope of this study.

IV. Algorithm

In this section, we introduce the bandwidth estimation algorithm. Our goal is to provide a tool that can estimate the available bandwidth of a mobile device by passively monitoring the traffic exchanged during the slow start phase of TCP, coping with all the artifacts described in Section III. The core idea can be summarized as: we try to identify groups of packets that arrive at the eNodeB together and thus were transmitted to the UE at the maximum speed that the scheduler could allocate at that instance. For each such group we compute the bandwidth by dividing the group's total number of bytes by its time duration. Because this approach is very susceptible to artifacts, we apply a set of filtering techniques before a result is derived.

We use TCPdump to sniff the traffic and organize it into flows based on IPs and ports. Packets related to TCP and TLS handshakes are ignored, since we are only interested in the data exchange part of the connection. On phones, most of the time only one TCP flow is actively downloading data [9], thus the chance of having overlapping flows is low. Even if there is an additional flow, it will most probably be in the slow start as well, or generate very low traffic. We may include these packets in the measurement, since our goal is to detect burst transmissions from the antenna.

An outline of the algorithm is presented in Algorithm 1 and its analytical presentation follows. For each incoming data packet P_i of a given flow, its size S_i and timestamp T_i are logged. The logging continues until a TCP FIN or RST packet is detected or until a few seconds have passed without any data exchange. The next step is to identify the groups of packets that were transmitted back-to-back from the server, which is done by locating unusually large delays between the arrivals of two consecutive packets. At this point, the timestamps of all the packets of the flow are available, so we derive their inter-arrival delays $D_i = T_{i+1} - T_i$. We ignore all the delays that are smaller than 1 ms, as such small delays imply that the packets were transmitted in the same TB and arrived at the same time at the NIC (i.e. packets that belong in the same "line", as shown in Figure 2a). The remaining delays are either caused by the scheduler organizing IP packets into consecutive TBs (delay between the last packet of a TB and the first packet of the next TB) or the server having paused the transmission because the congestion window limit has been reached (delay between the last packet of a server group and the first packet of the next server group). We want to identify the latter.

```
Data: Array $T$ of $n$ timestamps of packet arrivals
Array $S$ of sizes of $n$ packets
Result: Estimation of the instantaneous available
        bandwidth class
$D, G, BW \leftarrow \emptyset$;
for $i = 1$ to $n - 1$
do
    if $T_{i+1} - T_i \geq 1$ ms then
        $D = D \cup \{T_{i+1} - T_i\}$;
    end
end
$s \leftarrow 1$;
$P_1$ is the first packet of group $G_1$;
for $i = 1$ to $n - 1$
do
    if $T_{i+1} - T_i \geq \text{average}(D)$ then
        $P_i$ last packet of group $G_s$;
        $P_{i+1}$ first packet of group $G_{s+1}$;
        $s \leftarrow s + 1$;
    end
end
$P_n$ is the last packet of group $G_s$;
for $g$ in $G$
do
    if $T_{g_{last}} - T_{g_{first}} < 2$ ms then
        Delete $g$;
    else
        $BW = BW \cup \left\{ \frac{\sum S_g}{T_{g_{last}} - T_{g_{first}}} \right\}$;
    end
end
return $75^{th}$ percentile of $BW$;
```

Algorithm 1: Algorithm outline

Usually, a TB related delay is significantly smaller than a server group related one. However, this observation applies only to the early stages of slow start. For larger flows, which reach a state of almost continuous arrival of packets the two kinds of delays are indistinguishable. Such flows are beyond the scope of our tool, since other established bandwidth estimation tools can be used in case a flow is that large. If a delay D_i is higher than the average delay, we assume that it is server related, thus packet P_i is the last packet of the server group G_s and P_{i+1} is the first packet of the next server group G_{s+1}.

The duration of a group is the time difference between its first and its last packet. Each group that has a duration of less than 2 ms is ignored. In order to generate a bandwidth estimator from a group, it should consist of multiple TBs. Groups with a total duration of less than 2 ms usually are single-TB groups. For such, the measured time duration is unreliable since the packets arrive at the same time at the physical layer. At the kernel level where they are reported, there is a small time difference, which, if used to compute an estimator, would yield very high values. Also, such groups can be indicative of artifacts from a weak CPU, where a lot of packets appear at once after a long period of inactivity. The upper part of Figure 3 presents the first 100 packets of a high speed download generated by the Speedtest application. Our algorithm was able to identify six valid groups (marked between vertical lines), and one group that had to be ignored because of its small duration.

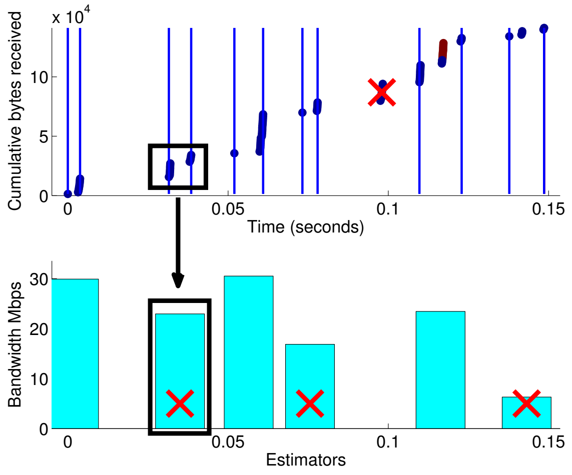


Fig. 3: First 100 packets of a TCP flow. The identified groups are enclosed between two vertical lines and their derived bandwidth estimators are located right below them.

For each remaining group g a bandwidth estimation value is derived by summing its total number of bytes over its duration:

$$BW_g = \frac{\sum S_g}{T_{g_{last}} - T_{g_{first}}} \tag{1}$$

We then filter out the lowest 50% of these values. Such small values are usually caused by smaller groups that are unable to fully utilize the available bandwidth (usually the very first server group, which was generated with the smallest possible congestion window value). Other possible reasons are sets of packets that arrived a little later at the antenna because of cross-traffic, or the algorithm splitting groups too aggressively. In order to avoid the effect of very large outliers, the median of the remaining samples is the output of the algorithm. These outliers could be caused by ordinary polling delays and weak hardware artifacts, which may significantly alter the timestamps. Effectively, these last two steps are implemented by picking the 75^{th} percentile of the values. In the example of Figure 3, the estimator values of each group are calculated in the lower part. The values of the 2nd, 4th and last group are ignored as part of the lowest 50% and the final result is the median of the remaining ones: about 30 Mbps.

This algorithm is so lightweight that it can be used on any modern mobile hardware with minimal impact on its resources. Not only is its complexity $O(n)$, but also n is bounded by the number of packets, which is at most a few hundred. The algorithm is designed to provide results for flows that have a number of packets that ranges from many tens (about 60-80) to a few hundred.

V. COMPARISON WITH BIN-BASED TOOLS

In this section we assess the accuracy of our algorithm. We collected traces from 4 European countries (Sweden, Germany, Spain, and Greece) using several FDD LTE devices. Our traces were collected by measuring traffic generated by automated tests and by volunteers who performed their usual tasks on our

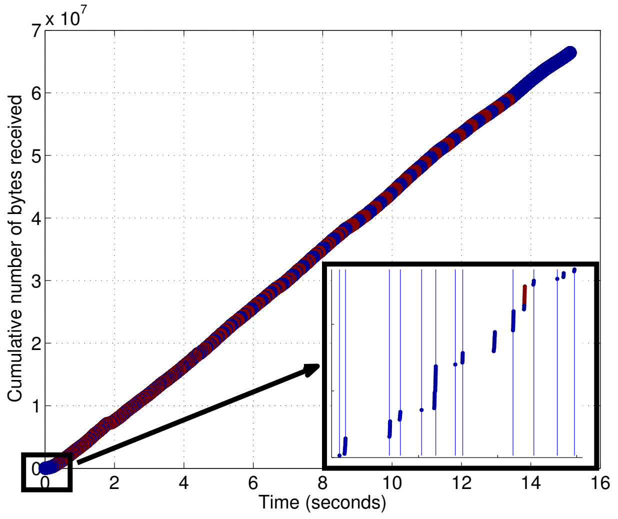


Fig. 4: A trace generated by the Speedtest application while the UE was stationary and connected to an uncongested cell.

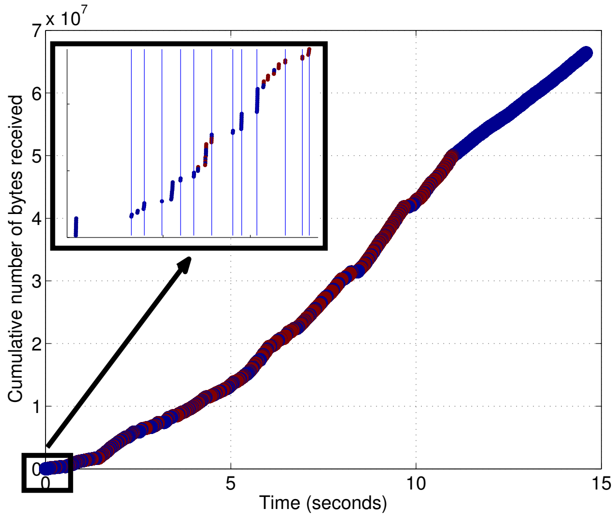


Fig. 5: A trace generated by the Speedtest application while the UE was in car moving with 100 Km/h.

instrumented devices. In this way, we can test our algorithm in a variety of scenarios and configurations.

To the best of our knowledge, there are no other bandwidth estimation tools that can exploit the traffic exchange of the slow start phase. An alternative would be to compare the resources allocated by the eNodeB scheduler to the result of our algorithm. This can be achieved by accessing the logs of an eNodeB in order to fetch the exact timing of each TB transmission. Obtaining such information is very hard, since it requires access to network components that are regarded as commercial secrets by both equipment vendors and mobile operators. Instead, we compare to a baseline bin-based algorithm similar to the one used in [19], [8]. If the measurement was generated by the Speedtest application, we use the result of this application instead. The baseline algorithm works as follows: 1) the data exchange part of a flow is isolated, then 2) the flow is split into 100ms bins, 3) a bandwidth estimation for each bin is generated by dividing the data exchanged by its duration, 4) the highest 10% and the lowest 30% of the values, as well as bins with no data, are discarded in order to reduce the effect of slow start and measurement artifacts and finally 5) the average of the remaining bins is returned. When the baseline algorithm is used on Speedtest generated traces, the deviation from the Speedtest value is at most 8%. Therefore, we believe its results are close to those of Speedtest. We remark that our algorithm only has access to a very small part of the data, while the baseline algorithm has access to the full trace.

A comparison between our algorithm and an active measurement one should be done with caution, because they are designed to compute different metrics. The data exchange part of files in the range of 100 KB, which is the target of our tool, requires no more than 500ms even in slow connections and can often be completed in less than 200ms. On the other hand, an active measurement tool requires very big downloads, that must be active for 10 to 20 seconds, in order to provide trustworthy results. Thus, our tool measures instantaneous bandwidth and an active tool measures the average channel capacity over the duration of the measurement. The perceived bandwidth at the side of the end user may greatly vary over a 10 second period. This can be caused by the channel experiencing fast variations (e.g., moving car), changing cell congestion, or packet loss. In such cases, it is expected that the results of the two tools differ. This effect can be seen in Figures 4 and 5, which present traces generated by the Speedtest application. Both traces, generate a similar pattern in their slow start phase (presented in the the smaller figures), which is what our tool is designed to use as input. The trace of Figure 4 continues to have a stable packet arrival pattern throughout its duration, in contrast to the one of Figure 5. Consequently, the deviation of the resulting speeds between the two tools is several times higher in the second case compared to the first. In addition, it is reported that even well regarded tools that are meant to measure speed on the less dynamic "wired scenario" using a similar approach may have significant deviation between each other [2].

For the rest of this section, the result generated by our tool is based on the first 100 packets of a flow. We use this limit in order to represent what it would be able to track if the flow was a short-lived TCP download that finished before exiting the slow start phase. Figure 6 results are generated by a small subset of the collected measurements generated solely by the Speedtest application. Except for some cases that were caused by the phenomena described above, the deviation is within 45%. Our tool is consistently giving lower values, because of the effect described in Figure 2d, since the speed achieved in most of the these traces, as reported by Speedtest, was mostly above 35Mbps.

Next, we filter our traces in order to keep flows that could be used by both our tool and the active bin-based estimator, described above. Thus, we reject flows that are too short, have too few packets, are UDP, use network technologies other than LTE etc. Consequently, the size of our sample reduces greatly from a few thousand flows to a few hundred, since most flows are too short-lived for the baseline algorithm. This further highlights the importance of a passive tool that can be used when very low traffic is present. The deviation between the values of the two tools is presented in Figure 7. For the majority of the cases the deviation is less than 50%. The traces which exhibit a deviation of more than 100% are caused mostly by flows that exhibit traffic patterns not suitable for binning (e.g., video streams) or extreme cases of channel variation (e.g., usage on a train). For example, a video stream is unsuitable for binning, because it is very bursty—there are long pauses and short bursts of rebuffering. The binning algorithm samples using a fixed time interval of 100 ms. Thus in every burst the first and the last bin are mostly empty, resulting in significant underestimation. Building a more robust binning for the baseline algorithm is beyond the scope of this work.

For the purpose of selecting the appropriate bit-rate for a streaming application, it is convenient to classify the resulting bandwidth into ranges. The number of classes and their limits are chosen with respect to bandwidth range categories that would make sense for a multimedia streaming application (bitrates of different stream qualities). Thus, the lower classes have significantly smaller range than the higher ones. To this end, we define five classes, that have the following Mbps ranges: 1) 0-5 Mbps 2) 5-10 Mbps 3) 10-20 Mbps 4) 20-60 Mbps and 5) higher than 60 Mbps. Table I presents how frequently our algorithm and the baseline match and by how many classes they differ, if they do not. We present both a comparison over all the traces and a comparison over only the traces that have traffic patterns more suitable for binning. Even though the two approaches have different objectives, in at least 60% of the cases they agree on the class. This is significant, considering that the result of our solution is derived with virtually no cost and there is no other tool that may provide such information.

Class difference	Same	1	2	3	4
Traces suitable for binning	70%	29.6%	0.4%	0%	0%
All traces	60.29%	30.88%	8.46%	0.37%	0%

TABLE I

Our approach is optimized for cellular scenarios but we have also performed some limited experimentation with WiFi. The challenges of a WiFi scenario are different from those of a cellular one. In brief, 1) the channel is half duplex, 2) usually a centralized scheduling entity is absent, 3) the significantly lower RTT times compared to cellular networks may make the detection of burst transmissions by the server harder, 4) the variable timing of the back-off mechanism, 5) the potentially weak hardware and the great variation of polling behavior, as presented in Section III-B and finally 6) the presence of broadcast traffic.

Despite the above, the same version of the algorithm, applied to a small WiFi trace was able to agree with the bin-based benchmark on the bandwidth class in at least 50% of the cases. Also, the deviation between the values of the two tools was again less than 50% in the majority of the trials.

VI. Discussion

This section highlights some important aspects of our solution. The small footprint of the algorithm makes it possible to run as a background service without influencing the OS's resource utilization. For example, it could be integrated in the interrupt or polling functions triggered upon packet arrivals. As mentioned in [7], some of the services that generate the flows we monitor might be rate limited on the server side or by a bottleneck link other than the antenna. In this case the tool does not measure the available bandwidth that the eNodeB

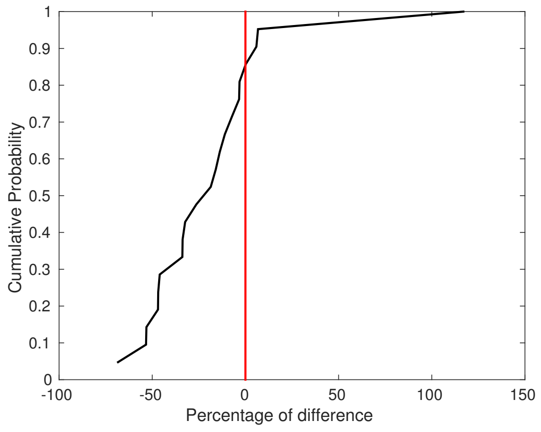


Fig. 6: Percentage of difference between the instantaneous bandwidth measured by our tool and the average bandwidth measured by the Speedtest APP.

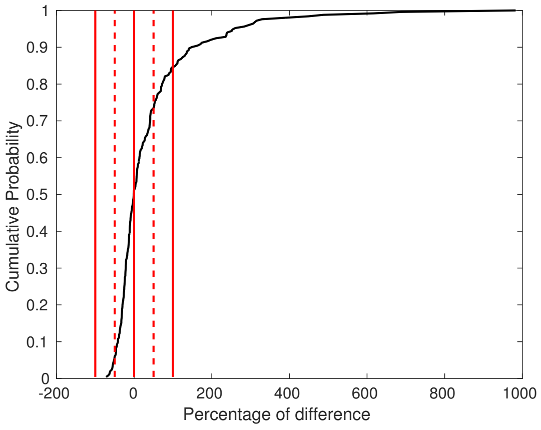


Fig. 7: Percentage of difference between the instantaneous bandwidth measured by our tool and the average bandwidth measured by a bin-based estimator. The solid and the dashed lines mark the 100% and 50% limits respectively.

can allocate to the user, but rather the rate of the server or the bottleneck link of the path. Thus, as with other passive techniques, the output of our algorithm should be seen as a lower bound. Also, a big portion of flows do not generate enough traffic to be used for a trustworthy estimation even for our tool. This includes applications that by definition use use very low traffic (less than 10 packets per 100ms), like chat applications (WhatsApp), or notification services like “Google cloud service”. Of course, if a user receives a media file in one of these apps, the traffic generated then is sufficient.

If the reported bandwidth is above 25Mbps, it is possible that the actual bandwidth that could be achieved by larger flows is significantly higher because of the phenomenon presented in Figure 2d. Also, it is possible that the slow start phase of TCP cannot (even momentarily) saturate the link enough for the estimators to get such high values. Our tool is meant to be used to optimize the QoE of media streaming applications and not to provide a highly accurate bandwidth estimation. An underestimation of the true value at such a high bandwidth class is not going affect the media application. Even the most demanding video streaming applications, like the Ultra

HD quality of Netflix require bandwidth in the range of 25 Mbps [18]. Thus, any value above that is guaranteed to ensure minimum start up delays and uninterrupted playback, while offering the best possible stream quality. If desirable, the exact value of bandwidth might be obtained by another solution after the playback has started. This tool is designed to operate during the slow start phase of a TCP connection. If the flow enters the steady state, our tool is under-performing, because it will try to find server related groups of packets, when the incoming packets form a continuous stream. In such cases, it is better to use another approach that is designed to work on large flows. Our tool is meant to be used as a complementary solution to the flows that can not be used by the existing bandwidth estimation algorithms.

Since our tool only tracks the size and time of the incoming packets, there is no privacy violation. The IP and port pairs are only used to identify when a flow is active and can be discarded after the measurement. Finally, when a user has a lot of small downloads within a short period of time, such as browsing web sites, the resulting estimations may be able to reflect the channel variation.

VII. Conclusions and future work

We have presented a very lightweight mobile bandwidth estimation tool that is able to provide reliable results by monitoring small data exchanges. To the best of our knowledge, it is the only tool that may provide an estimation relying solely on the traffic generated during the early phase of a TCP connection in mobile scenarios. Such flows are the vast majority in mobile networks. It is designed to be robust against various measurement artifacts introduced by the phone hardware and the scheduling process of mobile networks. It is ideal for enhancing the QoE of streaming applications. It can provide an estimation of the bandwidth available to a device, by just monitoring flows that precede a media streaming request, enabling the optimal selection of content bit-rate. Our solution is meant to be used alongside traditional bandwidth estimation tools that require access to large flows, thus offering reliable estimation for a great range of traffic. We have evaluated our approach with traces collected in 4 European countries with a variety of devices.

In the future, we plan to evaluate more precisely both the accuracy of our technique and the accuracy of network related information extracted at the kernel level of LTE phones. To do so, we intend to sniff the control channel of LTE and measure the delay between the arrival of a TB and the registration of the related IP packets at the kernel.

Disclaimer

We would like to emphasize that this is an academic research paper and should not be taken as indicative of Spotify's product plans.

Acknowledgments

This article is partially supported by the Madrid Regional Government through the TIGRE5-CM program (S2013/ICE-2919), the Spanish Ministry of Economy and Competitiveness through the Ramon y Cajal grant RYC-2012-10788 and grant TEC2014-55713-R, and from the European Union H2020-ICT grant 644399 (MONROE).

References

[1] H. Abou-zeid, H. Hassanein, and S. Valentin. Energy-efficient adaptive video transmission: Exploiting rate predictions in wireless networks. *IEEE Transactions on Vehicular Technology*, 63(5):2013–2026, June 2014.

[2] S. Bauer, D. D. Clark, and W. Lehr. Understanding broadband speed measurements. TPRC, 2010.

[3] N. Becker, A. Rizk, and M. Fidler. A measurement study on the application-level performance of LTE. In *IFIP Networking Conference*, pages 1–9, 2014.

[4] N. Bui and J. Widmer. Mobile network resource optimization under imperfect prediction. In *Proc. IEEE WoWMoM*, June 2015.

[5] S. Chen, S. Sun, Y. Wang, G. Xiao, and R. Tamrakar. A comprehensive survey of TDD-based mobile communication systems from TD-SCDMA 3G to TD-LTE(A) 4G and 5G directions. *Communications, China*, 12(2):40–60, Feb 2015.

[6] C. Dovrolis, P. Ramanathan, and D. Moore. Packet-dispersion techniques and a capacity-estimation methodology. *IEEE/ACM Transactions on Networking*, 12(6):963–977, December 2004.

[7] A. Gerber, J. Pang, O. Spatscheck, and S. Venkataraman. Speed testing without speed tests: estimating achievable download speed from passive measurements. In *ACM IMC*, pages 424–430, Melbourne, Australia, November 2010.

[8] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. A close examination of performance and power characteristics of 4G LTE networks. In *ACM MobiSys*, pages 225–238, Low Wood Bay, Lake District, United Kingdom, June 2012.

[9] J. Huang, F. Qian, Y. Guo, Y. Zhou, Q. Xu, Z. M. Mao, S. Sen, and O. Spatscheck. An in-depth study of LTE: Effect of network protocol and application behavior on performance. In *ACM SIGCOMM*, pages 363–374, Hong Kong, China, August 2013.

[10] C. Johnson. *Long Term Evolution IN BULLETS*. CreateSpace Independent Publishing Platform, 2 edition, 2012.

[11] R. Kapoor, L.-J. Chen, L. Lao, M. Gerla, and M. Sanadidi. CapProbe: a simple and accurate capacity estimation technique. *ACM SIGCOMM Computer Communication Review*, 34(4):67–78, October 2004.

[12] F. Kaup, F. Michelinakis, N. Bui, J. Widmer, K. Wac, and D. Hausheer. Behind the NAT – A measurement based evaluation of cellular service quality. In *CNSM*, 2015.

[13] G. Kreitz and F. Niemelä. Spotify – large scale, low latency, P2P music-on-demand streaming. In *Peer-to-Peer Computing*, pages 1–10. IEEE, 2010.

[14] K. Lai and M. Baker. Measuring link bandwidths using a deterministic model of packet delay. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ACM SIGCOMM '00, pages 283–294, New York, NY, USA.

[15] W. Li, R. K. Mok, D. Wu, and R. K. Chang. On the accuracy of smartphone-based mobile network measurement. In *IEEE INFOCOM*, Hong Kong, April 2015.

[16] Z. Lu and G. de Veciana. Optimizing stored video delivery for mobile networks: the value of knowing the future. In *IEEE INFOCOM 2013*, pages 2706–2714, 2013.

[17] F. Michelinakis, N. Bui, G. Fioravantti, J. Widmer, F. Kaup, and D. Hausheer. Lightweight mobile bandwidth availability measurement. In *IFIP Networking Conference*, May 2015.

[18] Netflix. https://help.netflix.com/en/node/306, last accessed June 2015.

[19] Ookla. Ookla speedtest mobile apps. http://www.speedtest.net/mobile/, last accessed June 2014.

[20] F. Ricciato, F. Vacirca, and M. Karner. Bottleneck Detection in UMTS via TCP Passive Monitoring: A Real Case. In *ACM CoNEXT*, pages 211–219, Toulouse, France, October 2005.

[21] TCPdump. http://www.TCPdump.org/, last accessed June 2015.

[22] Y. Xu, Z. Wang, W. K. Leong, and B. Leong. An end-to-end measurement study of modern cellular data networks. In *Passive and Active Measurement*, pages 34–45. Springer, 2014.