Throughput Estimation for Short Lived TCP Cubic Flows

Girisha De Silva, Mun Choon Chan and Wei Tsang Ooi
School of Computing, National University of Singapore
{girisha,chanmc,ooiwt}@comp.nus.edu.sg

ABSTRACT

Mobile devices are increasingly becoming the dominant device for Internet access. The network throughput achieved by a mobile device directly affects the performance and user experience. Throughput measurement techniques thus play an important role in predicting expected performance. Measurement techniques that require the transfer of large amounts of data can provide higher accuracy but incur large overhead. Further, since most mobile cellular plans impose usage quota, the overhead of such measurements over cellular networks can become quite high. Smaller data transfers have also been used to measure the throughput. Due to the conservative TCP slow start behaviour, however, these measurements often underestimate the achievable throughput. Considering these weaknesses in existing throughput measurement techniques, we propose a throughput estimation technique for TCP Cubic that uses 1 MB of data transfer to predict the throughput for prevalent large transfer sizes in mobile traffic such as 5 MB, 10 MB and 20 MB. Our evaluation shows that our approach can achieve high accuracy with low overhead, in predicting the achievable throughput.

CCS Concepts

•Networks → Network performance modeling; Network measurement; Mobile networks;

Keywords

Network Measurements, TCP Cubic, Throughput, Mobile Devices, Cellular Networks

1. INTRODUCTION

Mobile devices have overtaken desktops as the dominant device for Internet access [1, 2]. Therefore, the throughput a mobile device can achieve via different wireless technologies directly affects the performance and in turn the user experience.

Mobile device technologies are making massive advancements in terms of performance and features they provide. The wireless access technologies these devices use are also making massive strides in terms of the throughput they provide. For example, a recent study [3] reports that the median throughput for 4G LTE is 13 and 6 Mbps for downlink and uplink, respectively, and that these rates exceed the rates of WiFi and 3G. With these advancements, the traffic generated by mobile devices are also ever increasing. For example, a study conducted by ABI research in 2012 mentioned that the average smart phone app size is around 23 MB [4]. Apart from mobile apps, store and forward video services such as Vine and Instagram have also become popular. A study by Juniper states that these video files are 16 times larger than a photo and the estimate for a Instagram video is around 9.6 MB [5].

With the emergence of larger data transfers on mobile devices, estimating the throughput the mobile device can achieve via wireless access technologies is important towards providing the users with insights into the performance of their devices. Currently, two common ways of measuring the bandwidth are (1) transmit a large amount of data, say a few hundred MB to get a "stable" long term value, e.g., SpeedTest[1]; or (2) use a small amount of data, say 1 MB. Although transferring a large file will likely provide a more accurate result, the measurement overhead is high, needing close to 100 MB (see Section 3). Cellular data plans, however, often come with data caps. Three to four measurements of 100 MB each easily consume a significant portion of a 2 GB monthly data plan[2]. On the other hand, using a smaller data size to approximate the bandwidth directly would result in underestimating the available bandwidth by a great margin because of the transport level (TCP) behavior.

In this work, we aim to provide measurement results on LTE/WiFi bandwidth as well as highlighting the limitations of existing bandwidth measurements tools. Our contributions are: (1) provide measurement results of LTE and WiFi throughput over 10 different countries; (2) highlight the inaccuracy of using only a small amount of data to estimate bandwidth without taking into account the underlying TCP behavior; and (3) present a model for bandwidth estimation that works with small amount of data.

Our model is applicable to TCP Cubic. TCP Cubic is the current default TCP algorithm used by the Linux kernel [6]. A large chunk of publicly accessible servers in the Internet runs Linux. A measurement study carried out by W3Cook in May 2015 estimates that out of the top 1 millions websites (Alexa Rankings), 96% runs on the Linux operating system [7].

Our model is designed for data transfer sizes between 1 MB to 20 MB, the typical transfer sizes for applications such as app download and store-and-forward video. Our evaluation shows that the average error for bandwidth estimation using only 1 MB without taking into account the TCP behavior varies from 50% to 80% for transfer sizes of 5 MB to 20 MB. On the other hand our model has average errors of 8% to 11% for transfer sizes of 5 MB to 20 MB.

[1] http://www.speedtest.net/

[2] current default data cap for major telcos in Singapore

MOBIQUITOUS '16, November 28-December 01, 2016, Hiroshima, Japan

DOI: http://dx.doi.org/10.1145/2994374.2994391

Table 1: Categorization of throughput measurement techniques in existing works

Throughput Measurement Techniques	
Large Data Transfer	Small Data Transfer
[8, 9, 10, 11, 12]	[13, 14, 12]

The rest of the paper is organized as follows. Section 2 describes the previous research work related to modeling of TCP flows and also touches on previous throughput measurement studies in mobile networks. In Section 3, we present a detailed measurement study, in which we highlight the limitations of using smaller data sizes for measurements as well as highlight the high data overhead of applications such as SpeedTest. Section 4 describes our TCP Cubic modeling and throughput estimation technique in detail. Finally, in Section 5 we evaluate our throughput estimation technique.

2. RELATED WORK

2.1 Throughput Measurements

Many prior studies [8, 9, 10, 11, 12, 13, 14] have presented measurements of the delay and the available throughput in cellular and WiFi networks. The throughput is measured by either saturating the available channel capacity or by measuring the throughput achieved for smaller data transfers in the range of 1-2 MB. These studies also differ on the transport layer they used to carry out the measurement studies. Refer to Table 1 for a classification of different measurement techniques.

Yin Xu *et al.* [8] reported throughput measurements of cellular network providers in Singapore in which they used UDP flows to saturate the downlink mobile channel. On the other hand, the authors of [9] used TCP downlink and uplink flows to saturate the networks to measure the capacity of 3G networks while also measuring the effect on saturated TCP flows when constant bit rate applications such as voice or video calls are added to the network. Apart from [8, 9], many other measurement studies [10, 11, 12] also incorporate large data transfers and/or saturation of the data channel to infer the throughput of cellular/WiFi networks.

In order to saturate the link, the data sizes used range from few hundred MB to 1-2 GB (worst case) which is a large overhead given the relative small data quota for most cellular data plans that are typically in the range of a few GBs.

Studies such as [13, 14] used smaller data transfers to measure the throughput in mobile devices. Shuo Deng *et al.* [13] developed a mobile application that uses a 1 MB data transfer to measure the throughput on WiFi and cellular networks to enable the user to select the better network at a given time. Baranasuriya *et al.* [14] conducted a global cellular measurement study to evaluate the effectiveness of their bottleneck detector tool. They measured the TCP throughput of each of the network paths they studied with a data cap of 1 MB. Yung-Chih Chen *et al.* [12] used both small transfers and large transfers for their wireless multipath TCP performance evaluation.

In all the above measurements, they do not take into account the behavior of the underlying transport layer. Our measurements show that due to the conservative nature of TCP in the initial phase, usage of a small transfer size such as 1 MB significantly underestimate the achievable throughput for large transfers.

2.2 TCP Throughput Modeling

One of the earliest works in modeling TCP throughput can be found in [15]. The analytical model proposed is for TCP Reno and it models the steady state throughput of a TCP Reno flow taking into account loss and (average) round trip time (RTT). Another popular TCP variant deployed is TCP Cubic, the default choice for Linux based system. TCP Cubic uses slow start and a cubic function during the congestion avoidance phase. An analytical model for the steady state throughput of a single TCP Cubic flow can be found in [16]. A Markovian model is incorporated in modeling the TCP Cubic flow, assuming that the RTT is constant and the packet loss rate is Poisson. Another analytical model proposed for TCP Cubic steady state throughput can be found in [17]. Unlike the model proposed in [16] the analytical model in [17] takes into account changes in RTT values by estimating the average RTT using the sojurn time of M/GI/1 queues. Both the solutions [16, 17] model the steady state of TCP Cubic with packet losses and was evaluated using network simulators.

A model for short lived TCP flows by taking into account the slow start phase has been introduced by Marco Mellia *et al.* [18]. In their model, however, the short lived TCP flow never exits the slow start phase. [19, 20] also model short lived TCP flows. In these works, the modeling is done for TCP Reno. To the best of our knowledge, there are no throughput estimators for short TCP Cubic flows that incorporates the behavior of the *HyStart* [21] slow start algorithm. Refer to Table 2 for a classification of TCP throughput modeling.

Table 2: Categorization of TCP throughput modeling. Steady state models are based only on the congestion avoidance phase. Short lived flows model slow start and/or congestion avoidance

Throughput Models for TCP Flows	
Steady State	Short Lived
[15, 17, 16]	[18, 19, 20]

Our work is different in that we are looking at throughput estimation that uses only a small amount of data (1 MB) but can be used to estimate the achievable throughput for much larger data transfers such as 5 MB, 10 MB, and 20 MB.

3. MEASUREMENT STUDY

3.1 Measurement Tool

We carried out a measurement study using a tool built in the form of an Android application. The user interface of our application is shown in Figure 1.

The application's main functionality is to run a throughput measurement and a delay measurement. The work flow of the application is as follows:

1. The user selects whether to run the measurement over WiFi or Cellular.
2. Enter in a short descriptive tag about the location (e.g., Home, Office)
3. Pick the measurement type
   (a) Delay Measurement
   (b) Throughput Measurement (Labeled as Data on Figure 1)
      i. Pick a data size to measure throughput
4. Press start and wait for the measurement completion

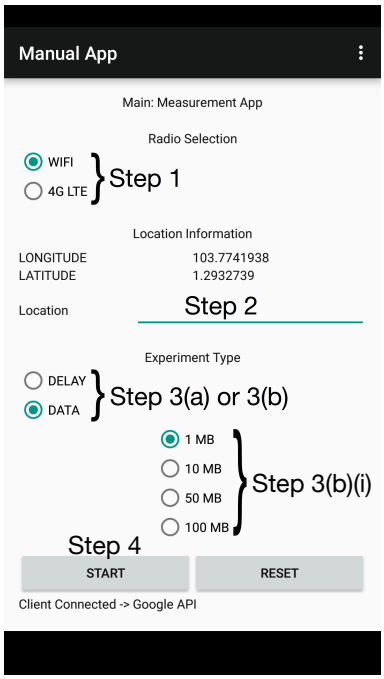
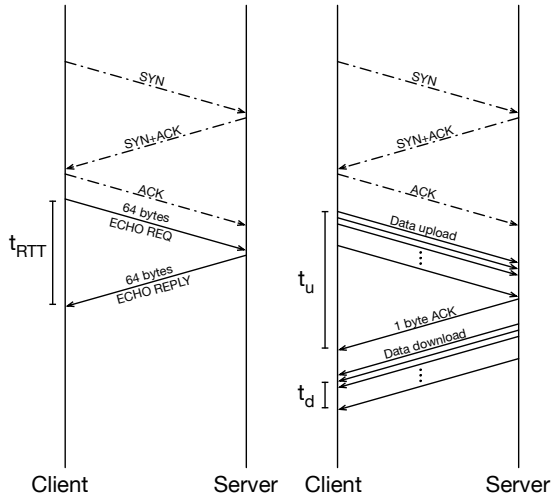


Figure 1: The main UI of the Android Application used for measurement



(a) Delay Measurement (b) Throughput Measurement

Figure 2: The two experiments carried out for measuring the delay and throughput in cellular and WiFi networks

3.1.1 *Measurement Types*

Delay Measurement: The mobile application establishes a TCP connection with the designated server and sends 64 bytes that is echoed back to the client by the server. The client keeps track of the total round trip time (t_{RTT}) to send and then receive 64 bytes. (Refer Figure 2a)

Throughput Measurement: Once the user selects a data size, the mobile application establishes a TCP connection with the server. The selected data size is then sent (uploaded) to the server. Once the server receives all the data, it sends a one byte ack to the client. The total time that elapsed from uploading the data to receiving the one byte ack is recorded by the client as the upload time (t_u).

As soon as the one byte ack is sent, the server sends the selected amount of bytes to the client. The client then starts a timer as soon as the first byte(s) are received from the server and stops it after all the bytes have been received. This timer value is considered as the download time (t_d). The client keeps track of the total amount of data received during the t_d time period. (Refer Figure 2b)

3.1.2 *Measurement Setup*

For the purpose of delay and throughput measurements, we have selected a server running an instance of Ubuntu 64 bit (Kernel ver. 2.6.32). The server resides in Dallas, Texas, USA and two Java server programs listened on port 80 and port 443 respectively. The client application connects to port 443 for delay experiments and connects to port 80 for throughput measurements.

For every throughput and delay measurement, we used the same server in Dallas irrespective of which country we were doing the measurement from. The client application (see Section 3.1) always runs on an Android mobile device with the exception of when cellular data was collected in India. Out of the 476 cellular throughput samples (refer Table 3) collected in India, 464 samples were collected using a 4G dongle connected to a Windows 7 laptop. A Java based client program was used in this instance. In this measurement study, we do not differentiate between the cellular technologies such as LTE, HSPA, 3G, or GSM.

3.2 Measurement Results

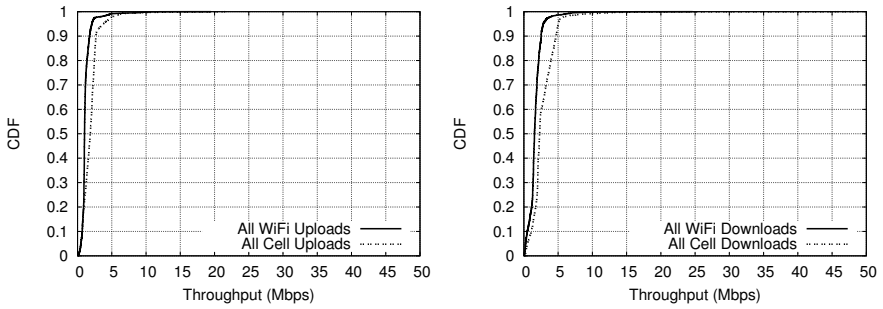
We have collected throughput measurement data from 10 different countries. In total, we have collected 8273 WiFi throughput samples and 3797 cellular throughput samples. Refer to Table 3 for a summary of the measurements collected.

Table 3: Number of total WiFi and Cellular throughput measurements carried out in different geographic regions. The last column represents the number of unique cities the measurements covered in each respective country

Country/Region	WiFi	Cellular	Cities
Singapore	7666	2252	1
USA	62	104	5
South Korea	117	56	2
Sri Lanka	277	547	2
India	57	476	2
Australia	-	248	2
England	51	53	1
Taiwan	25	55	1
Hong Kong	18	-	1
Thailand	-	6	1

3.2.1 *WiFi vs. Cellular Throughput*

Figures 3a and 3b show the CDF for both WiFi and cellular upload (uplink) and download (downlink) throughput measurements for all the data sizes collected in countries listed in Table 3. In the case of downlink throughput (Figure 3b), cellular networks outperform WiFi 95% of the time. The 90th percentile for Cellular and WiFi throughput are 4.7 Mbps and 2.5 Mbps, respectively. Cellular networks outperform WiFi 75% of the time for uplink throughput (see Figure 3a), and the 90th percentile for cellular and WiFi is 2.69 Mbps and 1.76 Mbps, respectively.



(a) Cellular vs. WiFi Upload Throughput (b) Cellular vs. WiFi Download Throughput

Figure 3: Cellular vs. WiFi upload and download throughput measurements for all transfer sizes

Previous studies have reported that LTE networks outperform WiFi networks [3, 13] in terms of downlink and uplink throughput. In our measurement study, out of the 3797 cellular throughput samples collected, 2648 samples are from LTE networks (70% of the data). Our results are consistent with previous findings.

3.2.2 *Throughput: Small Transfers vs. Large Transfers*

Figure 4 shows the measured uplink and downlink throughput distribution for all countries mentioned in Table 3. In these figures, the measured throughputs are shown for small (1 MB) and large ($\geq$ 10 MB)

In the case of cellular and WiFi, it can be seen that the uplink and downlink throughput measurements using larger data transfers shows significantly higher values compared to measurements using only 1 MB data transfers. The measured average for cellular and WiFi uplink throughput when 1 MB of data was used was 1.81 and 1.41 Mbps, respectively. Further, when 1 MB of data was used to measure the downlink throughput the average values were 2.65 and 1.62 Mbps for cellular and WiFi, respectively.

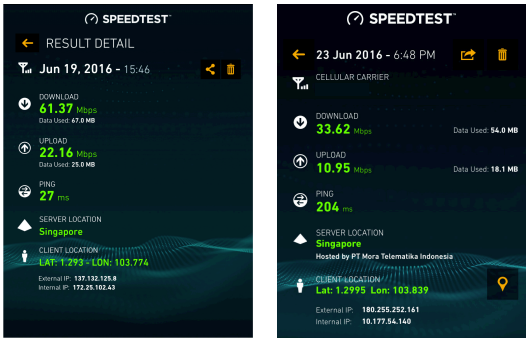
In contrast, when transfer sizes larger than or equal to 10 MB were used for measuring the throughput, the average uplink throughputs for cellular and WiFi are 7.97 and 6.15 Mbps; the average downlink throughputs are 10.24 and 5.65 Mbps. The throughputs of large transfers are thus significantly higher than 1 MB transfers by 340% and 286% for cellular uplink and downlink respectively. For WiFi, the average throughput values for large transfers are significantly higher than 1 MB transfers by 330% and 249% for uplink and downlink respectively.

There is a clear difference between the measured throughput values obtained with smaller data transfers and larger data transfers. While, it is expected that measurements using only 1 MB tends to underestimate the achievable throughput, the large difference in results brings to question the utility of using only a small amount of data transfer to estimate achievable throughput. The main reason behind this large deviation is the behaviour of the congestion control algorithm of a TCP flow. The congestion control algorithm of a TCP flow controls the amount of data it can send at a given time, which in turn directly affects the throughput a TCP flow could achieve. The main emphasis of the congestion control algorithm is to probe the available bandwidth in the network while being fair to the existing flows. The algorithm initially enters a phase named the slow start phase where the amount of segments it sends grows exponentially every RTT. Although the growth is exponential during the slow start phase, the TCP flow will initially send small number of segments in order to be conservative when probing the available bandwidth. Once the slow start phase ends, the TCP flow will enter a congestion avoidance phase where it will still keep probing for the available bandwidth using a window growth function that depends on the TCP variant that is being used.

If smaller data transfers are used for measuring the throughput, most often these flows might not have achieved the total available bandwidth due to the initial conservative approach of the TCP congestion control algorithm. Thus, a large data transfer is often used to get a better estimate of the available bandwidth.

3.3 SpeedTest Results

We have also carried out measurements using the SpeedTest application downloaded from the Google Play Store (Android)[3] and the Apple App Store (iOS)[4]. After the measurements were conducted, we checked the total data usage of the app using the phones inbuilt software that keeps track of the data usage by individual applications. The reported total usage for two distinct Android devices were 92 MB (see Figure 5a) and 125 MB. In case of iOS the total data usage for two different iPhones were 72.1 MB (see Figure 5b) and 82.4 MB respectively.

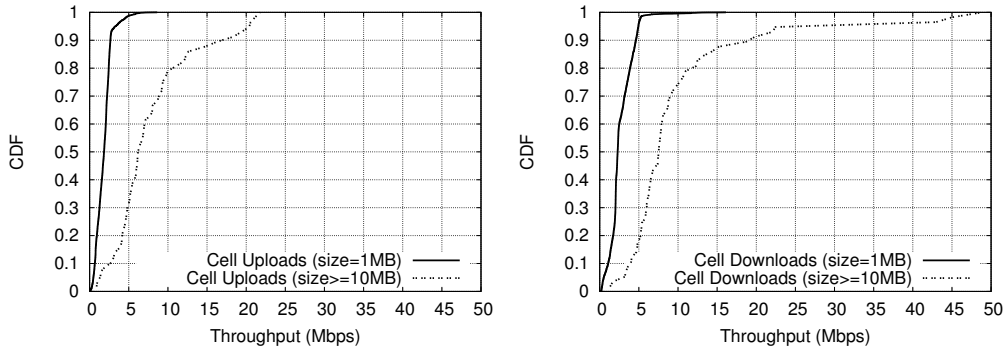


(a) Android (b) iPhone

Figure 5: Statistics reported by SpeedTest on Android and iOS. The application reports the delay, uplink and downlink throughput and also the amount of data used for a single measurement

Mobile cellular plans are usually capped by the service providers. From our experiments, a single SpeedTest experiment could potentially (based on our results) cost up to 4% of the total data available in a 2 GB monthly data plan, thus incurring a significant overhead

[3]https://goo.gl/gtXLBc
[4]https://goo.gl/WUXJQx



(a) Cellular Upload Throughput

(b) Cellular Download Throughput

(c) WiFi Upload Throughput

(d) WiFi Download Throughput

Figure 4: Throughput measurements for small (size = 1 MB) data transfers and large (size $\geq$ 10 MB) data transfers in WiFi and Cellular Networks

for bandwidth measurements.

3.4 Data Download Experiment

In this measurement, we look at how the throughput measurements vary when different data transfer sizes are used. The measurement is performed in the following. A client program running on an Android phone downloads 20 MB of data from the server. The program keeps track of the time it has received 1 MB, 5 MB, 10 MB, and 20 MB. Table 4 shows the throughput measurements for three such download sessions. One can observe that the throughput increases when the amount of data transferred increases. In particular, if one simply downloads 1 MB and uses the observed throughput to estimate the throughput for large data sizes, the error is large, ranging from 20% to 85%.

When averaged over 20 such experiments, the average deviation for throughputs between the pairs (1 MB, 5 MB), (1 MB, 10 MB), and (1 MB, 20 MB) are 54%, 67% and 77%, respectively.

This observation motivates us to ask the following question. Is it possible to have a more accurate estimate of the throughput for higher data transfer sizes ($\geq$ 5 MB) and yet only use a 1 MB data transfer in order to reduce the measurement overhead? In order to achieve this objective, we present in the next section, a simple model for throughput estimation for short lived TCP Cubic flows.

4. MODELLING TCP CUBIC

TCP Cubic has been the default TCP variant supported in the Linux kernel since version 2.6.13 [6]. TCP Cubic, which consists of a cubic window growth function, was introduced to achieve better performance over networks with higher bandwidth and longer delays. In this section, we provide a brief overview of TCP Cubic. More information can be found in [6, 21].

Table 4: Measured throughput values for 1, 5, 10, and 20 MB when downloading 20 MB of data. For each download session, the table also states the deviation (d) if the throughput for 5, 10, and 20 MB were based on the throughput measured for 1 MB

	Throughput (Mbps)			
	1 MB	5 MB (d)	10 MB (d)	20 MB (d)
Session 1	5.51	14.37 (62%)	18.21 (70%)	21.48 (74%)
Session 2	1.69	4.34 (61%)	7.07 (76%)	11.55 (85%)
Session 3	3.79	4.71 (20%)	5.75 (34%)	8.36 (55%)

4.1 TCP Cubic

4.1.1 Slow Start

The standard slow start phase of TCP uses an exponential congestion window growth scheme, i.e., doubling the window every round trip time. As the exponential growth of the congestion window could result in a large number of packet losses, a hybrid slow start scheme is introduced in TCP Cubic.

The hybrid slow start scheme *HyStart* [21] uses the same exponential window growth during slow start with the exception of dynamically setting the slow start threshold (*ssthresh*). The setting of the dynamic *ssthresh* value is based on two main mechanisms

running in parallel. The first mechanism uses an ACK train generated from the TCP receiver to estimate whether the forward path is congested and the other mechanism uses RTT delay samples to gauge whether the flow has reached the capacity of the forward path. If any one of the two mechanisms detects that the exponential slow start growth needs to end, then *ssthresh* is set to the current window size and TCP Cubic enters the congestion avoidance phase where it follows a cubic window growth function.

4.1.2 Congestion Avoidance

The congestion avoidance phase in TCP Cubic calculates the congestion window, $W(t)$, every RTT using Equation 1 where W_{max} is the congestion window when the last loss occurred, t is the elapsed time since the last loss event, K is time it takes the congestion window to increase from the last reduction to W_{max} if no further packet loss happens and C is the cubic parameter.

$$W(t) = C(t-K)^3 + W_{max} \tag{1}$$

As the congestion window growth is based on a cubic function, the window growth has two distinct regions: 1) the concave region ($W(t) < W_{max}$); 2) the convex region ($W(t) > W_{max}$). Initially the congestion window grows rapidly in the concave region but slows down and plateaus at W_{max}. If no further loss occurs, the congestion window growth function will next enter the convex region where initially the congestion window growth is slow when the congestion window value is near W_{max} but the growth rate will increase with the congestion window value moving further away from W_{max}.

W_{max} is the value in which the last packet loss occurred and a packet loss event most often signifies that a TCP flow has reached the current available capacity. Thus the concave region of the cubic function grows the window rapidly, but slows down and becomes cautious when the congestion window value approaches W_{max}. If no further loss occurs around W_{max}, there is more available capacity for the TCP flow and the convex region will rapidly start to increase the congestion window to find the new W_{max} value i.e., capacity of the channel after being cautious near W_{max}.

One exception to the cubic growth rate is performed when $W(t)$ is found to be growing slower than TCP Reno. When this occurs, TCP Cubic is set to work in the "TCP friendly region" whereby the window size is set to be the same as TCP Reno.

4.2 Modelling Short TCP Cubic Flows

In this section, we present our model of a single short lived TCP Cubic flow which we will use in our throughput estimation later. In order to be useful, the model derived should be able to perform the following. Given (measurable) quantities such as estimated round trip time, data size transfer (X), and data transfer time, the model provides an estimate of the time it takes to transfer Y (where $Y > X$) bytes.

The model assumes that the TCP Cubic flow begins from the slow start phase and then enters the congestion avoidance phase after some number of rounds have passed using the *HyStart* algorithm [21]. In this model, we assume that RTT is constant and there is no packet loss.

Figure 6 represents a typical window growth scenario for a TCP Cubic flow with no loss. The initial window size for the TCP flow is W_0 and the flow is in slow start phase until the x^{th} transmission round. The congestion window value when TCP Cubic quits slow start is W_s.

Initially, the slow start phase in TCP Cubic transmits W_0 segments. Once the first ACK is received, the congestion window is increased by 1. Since the congestion window was increased by 1

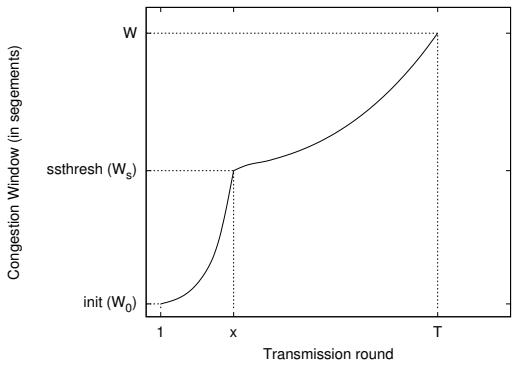


Figure 6: TCP Cubic: Slow Start + Congestion Avoidance

and one packet has already being ACKed, the congestion control algorithm can now transmit two packets. Thus, during the slow start phase for every ACK received, two more packets are transmitted. Moreover, the total number of ACKs received when the congestion window is W_s can be expressed as $(W_s - W_0)$. Since each ACK triggers two transmissions, the total number of segments transmitted during the slow start phase can be expressed as 2.

$$2(W_s - W_0) + W_0 \tag{2}$$

Once the TCP Cubic flow exits the slow start phase, it enters the congestion avoidance phase which uses the cubic equation given in Equation 1 to increase its window size every RTT. The window growth function given in Equation 1 can be simplified as $W(t) = C(t)^3 + W_s$. This simplification is derived from the source code of TCP Cubic [22] and it can be intuitively explained as follows. After slow start, TCP Cubic sets the current ssthresh (W_s) as W_{max} and the cubic function is in the convex region (i.e., the cubic function has already plateaued at W_s). Further, since the function has already reached the plateau (W_{max}), $K = 0$.

Using this simplification, the total number of segments transmitted till the T^{th} transmission round during congestion avoidance can be expressed using Equation 3.

$$(W_s + C(RTT)^3) + (W_s + C(2RTT)^3) + ... + (W_s + C((T-x)RTT)^3) \tag{3}$$

Since, $1^3 + 2^3 + 3^3 + ... + n^3 = \frac{n^2(n+1)^2}{4}$, we can further simplify Equation 3 to the following:

$$(T-x)W_s + \frac{C(T-x)^2(T-x+1)^2RTT^3}{4} \tag{4}$$

Assuming that there is no packet loss, the total number of segments, N, transmitted during T transmission rounds can be expressed in Equation 5 using Equations 2 and 4.

$$(2+T-x)W_s - W_0 + \frac{C(T-x)^2(T-x+1)^2RTT^3}{4} = N \tag{5}$$

Assuming that RTT is constant, given the total time ($total\_time$) to receive N segments, the total number of rounds, T, can be derived from $T = total\_time/RTT$. Further, the number of transmitted segments, N, can be derived by $N = total\_data\_sent/MSS$, where MSS is the maximum segment size.

Note that there are two more unknown variables left in Equation 5, the values for the transmission round (x) and the congestion window value (W_s) when the TCP Cubic flow quits slow start. In Section 4.2.1, we describe a simple iterative method which derives the values for x and W_s using Equation 5.

4.2.1 Throughput prediction for short TCP Cubic flows

Algorithm 1, the *HyStart* solver, is an iterative solver for Equation 5. The solver takes in as input the total time ($total\_time$) for completion of the data transfer, the round trip time between the two communication nodes (RTT) and the amount of data transferred ($data$).

The algorithm first derives the total number of rounds (T) to transfer the data. Since we assume that RTT is constant, the total number of rounds is derived by dividing the total time by the RTT. Once the total number of rounds are known, the solver iterates through valid W_s values and uses Equation 5 to calculate the total number of segments transferred. The first instance of W_s that yields the total number of segments larger than or equal to to the total segments transferred (N) is returned as the solution. The total segments transferred, (N), is derived by dividing the total data transferred by the maximum segment size (MSS).

Algorithm 1 *HyStart* Solver

Input:
$RTT \leftarrow$ round trip time
$data \leftarrow$ the total transfer size
$total\_time \leftarrow$ total time for the data transfer
Output:
$W_s \leftarrow$ ssthresh

1: $C \leftarrow$ cubic constant
2: $MSS \leftarrow$ maximum segment size
3: $W_0 \leftarrow$ initial TCP window (constant)

4: $W_s = null$
5: $N = \lceil data/MSS \rceil$
6: $T = (total\_time)/RTT$

7: **for** $r = 1; r{+}{+}$; while $r \leq T$ **do**
8: **for** $w = 2^{r-1}W_0; r{+}{+}$; while $r < 2^x W_0$ **do**
9: $ss\_seg = 2w - W_0$
10: $ca\_seg = (T - r)w + C(T - r)^2(T - r + 1)^2(RTT)^3(1/4)$

11: **if** $w > N$ **then**
12: *break*
13: **end if**

14: $total\_seg = ss\_seg + ca\_seg$

15: **if** $total\_seg \geq N$ **then**
16: $W_s = w$
17: return W_s
18: **end if**

19: **end for**
20: **end for**

Once the slow start threshold W_s is derived using the *HyStart* solver (see Algorithm 1), throughput for a given size can be calculated using the estimator algorithm given in Algorithm 2.

The estimator algorithm (see Algorithm 2) models the behaviour of a TCP Cubic flow consisting of a slow start phase and a congestion avoidance phase. The algorithm takes in as input the size for which the throughput needs to be estimated, the RTT between the communication end points, and the slow start threshold (*ssthresh* W_s). Using the given input size, the algorithm calculates the total segments needed to be transferred. Further, the total number of segments transferred in the slow start phase and the number of rounds the slow start phase operated is calculated based on the *ssthresh* value provided. Since the total number of segments to be transferred is known, the algorithm iterates through the congestion avoidance phase until the necessary number of segments have been transferred.

Note that the size of the congestion window is limited by the receiver buffer size. This buffer limit is device and network specific [23].

With Algorithms 1 and 2, given the estimated RTT, a small transfer data size of say 1 MB and the transfer time, we are able to estimate the transfer time for larger data sizes. As Algorithm 1 iterates over the number of rounds and Algorithm 2 iterates over the number of segments transferred, the computation overheads for both algorithms are minimum.

Algorithm 2 Throughput Estimator

Input:
$size \leftarrow$ size for throughput prediction
$RTT \leftarrow$ round trip time
$W_s \leftarrow$ ssthresh
Output:
$eT \leftarrow$ estimated throughput

1: $clamped \leftarrow False$
2: $C \leftarrow$ cubic constant
3: $MSS \leftarrow$ maximum segment size
4: $W_0 \leftarrow$ initial TCP window
5: $cw\_clamp \leftarrow$ cong. window clamp

6: $t = 1$
7: $eT = 0$
8: $ca\_seg = 0$
9: $total\_seg = \lceil size/MSS \rceil$
10: $ss\_seg = 2W_s - W_0$
11: $x = \lfloor log_2(W_s/W_0) + 1 \rfloor$

12: **while** True **do**
13: **if** not $clamped$ **then**
14: $cwnd = W_s + C(tRTT)^3$
15: **if** $cwnd \geq cw\_clamp$ **then**
16: $cwnd = cw\_clamp$
17: $clamped = True$
18: **end if**
19: **end if**
20: $ca\_seg = ca\_seg + cwnd$
21: **if** $ca\_seg + ss\_sg \geq total\_seg$ **then**
22: $eT = size/(RTT(t + x))$
23: **return** eT
24: **end if**
25: $t = t + 1$
26: **end while**

5. EVALUATION

5.1 Evaluation Setup

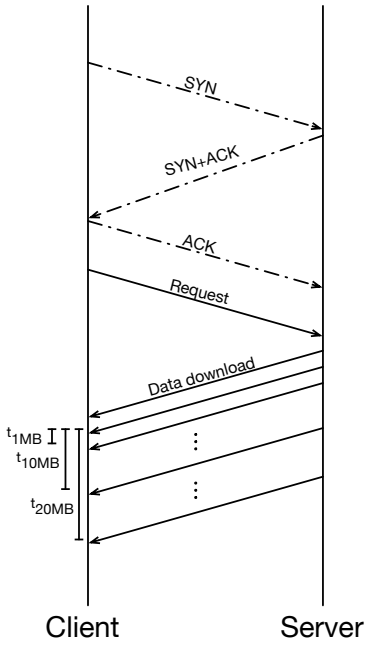


Figure 7: The experiment for evaluating the throughput estimation model

We used an Amazon AWS EC2 instance running an Ubuntu 64 bit Linux (Kernel ver. 3.13) as the server. The server was placed in the west coast of USA and it consisted of two Java server programs that were listening on port 80 and 443 respectively. Further, the server also consisted of a modified version[5] of TCP_PROBE [24] that was logging the TCP state for flows connecting to port 80. We used TCP_PROBE mainly to log (monitor) the congestion window evolution when the server is transmitting data to a client connected on port 80. The server also ran the default version of TCP Cubic that comes with the Linux kernel version 3.13.

The client, an Android application, initially carries out two delay measurements by connecting to port 443 on the server. The delay measurement is identical to the one described in Section 3.1.1. Once the two delay measurements are carried out the client then sends a request to the server on port 80 that triggers the server to send 20 MB of data. Just as the download throughput experiment described in Section 3.1.1, the client starts a timer once the first byte(s) are received from the server. Further, the client keeps track of the bytes being received so that the it can track the time to receive 1 MB, 5 MB, and 10 MB in a 20 MB download (see Figure 7). The end of the download triggers the client to carry out three more delay measurements on port 443. For all 20 MB download sessions we used a LG Nexus 5 running Android version 6.0.1 (Marshmallow). The receiver buffer size for 4G LTE networks for this mobile device was 1048576 bytes. This value is specified in the *init.rc* file that can be found in the mobile device [23]. Based on this buffer size we estimated the congestion window growth to be clamped when it hits the value of 800. Further, using a TCP dump trace, we estimated the maximum segment size (MSS) between the client and server to be 1388 bytes.

[5]https://goo.gl/K3YOro

Once a single download session is completed, we first average the five delay samples and use this value as the round trip time (RTT) between the client and the server for the session. Further, we derive the total number of transmission rounds to transfer the first 1 MB of data by dividing the total time to receive the first 1 MB (out of 20 MB) of data by the estimated RTT. With these values, we used the *HyStart* solver (Algorithm 1) to estimate the congestion window value, in which the TCP flow quits slow start (*ssthresh*). Finally, using the estimated *sshthresh* value we estimated the throughput for 5 MB, 10 MB, and 20 MB using the *Throughput Estimator* (Algorithm 2).

5.2 Results

We collected 58 samples of 20 MB download sessions using a 4G LTE connection in Singapore. For each sample collected, the measured throughput values for 1, 5, 10, and 20 MB downloads were recorded. We then estimated the throughput values for 5, 10, and 20 MB downloads using the proposed throughput estimator.

For each size, 5, 10, and 20 MB, we calculate the error when estimating the throughput via the proposed estimator and also when using the 1 MB throughput measurement as the estimator. The results are shown in Figure 8.

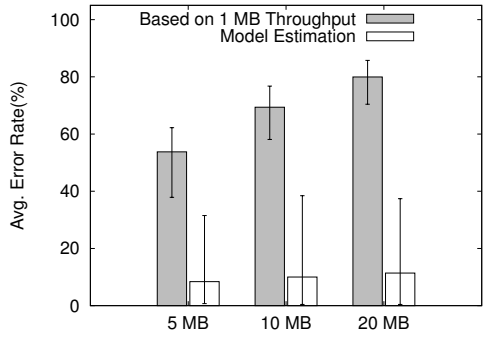


Figure 8: Average error rate when estimating the throughput via the proposed estimator and when using the measurement obtained for a 1 MB data transfer. The error bars represent the 90^{th} and the 10^{th} percentiles

The average errors when using the 1 MB measurement as the estimated values are 53.77% (10^{th} percentile: 37.88%, 90^{th} percentile: 62.22%), 69.38% (10^{th} percentile: 58.11%, 90^{th} percentile: 76.76%) and 79.95% (10^{th} percentile: 70.41%, 90^{th} percentile: 85.76%) for 5, 10 and 20 MB respectively. On the other hand, the average errors for the values obtained from the throughput estimator are 8.41% (10^{th} percentile: 0.71%, 90^{th} percentile: 31.5%) 10.01% (10^{th} percentile: 0.41%, 90^{th} percentile: 38.45%) and 11.39% (10^{th} percentile: 0.43%, 90^{th} percentile: 37.41%) for 5, 10 and 20 MB respectively.

Based on the TCP_PROBE logs, 41 samples out of the 58 collected have no congestion window reduction due to packet loss in the congestion avoidance phase. The average errors for these 41 samples are given in Table 5. A comparison between the estimated congestion window evolution and the logged congestion window evolution for a single download session can be found in Figure 9a. In these cases, it can be seen that the proposed model is accurate. The estimated throughput estimation errors for the scenario in Figure 9a are 1.61%, 1.84%, and 0.52% for 5, 10, and 20 MB respectively.

There are 13 cases (out of 58) with time-out and fast recovery events. The average errors are given in Table 6. One scenario where

Table 5: Average error rates for the proposed estimator and when using the 1 MB measurement as throughput. Contains samples that had no reduction in the congestion window (due to packet loss) during the congestion avoidance phase. The values in brackets represents the standard deviation (std)

Download Size	Error Rate: Estimator	Error Rate: 1 MB
5 MB	3.22% (std:2.93%)	54.87% (std:8.59%)
10 MB	2.42% (std:2.35%)	71.28% (std:7.22%)
20 MB	2.02% (std:1.69%)	82.14% (std:5.06%)

a fast recovery event occurred is shown in Figure 9b. In this case, the errors for the estimated throughput values for 5, 10, and 20 MB are 0.7%, 10.72%, and 35.38%, respectively. If the 1 MB throughput measurement is used as the estimator instead, the error for the 20 MB download is as high as 80%.

Table 6: Average error rates for the proposed estimator and when using the 1 MB measurement as throughput. Contains samples which had packet loss. The values in brackets represents the standard deviation (std)

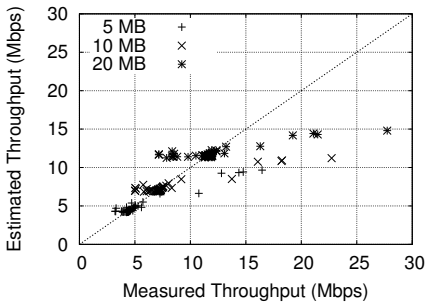
Download Size	Error Rate: Estimator	Error Rate: 1 MB
5 MB	20.91% (std:14.79%)	51.11% (std:13.38%)
10 MB	28.30% (std:15.40%)	64.79% (std:11.55%)
20 MB	33.99% (std:15.97%)	74.67% (std:7.96%)

We observed that in a small number of cases, the entire 1MB was transferred in the slow start phase. A sample congestion window evolution for this scenario can be found in Figure 9c. In our model, we will simply assume that the *ssthresh* occurs at the 1MB transfer. In these cases, the transfer size of 1MB is simply too small to provide sufficient information even for our model to estimate the transition point between slow start and cubic growth. For these 4 cases, the average error rates are 35%, 41%, and 34.5% for 5, 10, and 20 MB respectively when the proposed estimator is used. If the 1 MB throughput is used instead, the average errors are as high as 60.66%, 69.25%, and 74.1% for 5, 10, and 20 MB, respectively.

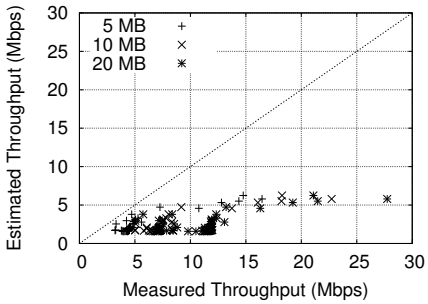
Figure 10 shows the scatter plots of the measured vs. the estimated throughput values using the proposed estimator and the 1 MB based throughput estimator. We observe that the proposed estimator performs well for throughput values up to about 13 Mbps. Beyond that, throughput estimates plateau at around 15 Mbps, showing the limits of using only 1 MB for measurements. From Figure 4, 15 Mbps is the 90th percentile for both cellular upload and download. On the other hand, estimation error is high if 1 MB data transfer throughput is used to estimate higher throughput. In fact, the throughput estimates plateau around 5 Mbps. From Figure 4, 5 Mbps is the 30th percentile for cellular upload and 20th percentile for cellular download. Note that the estimated throughput by our estimator shown in Figure 10a is clustered around 4.5 Mbps, 7 Mbps and 12 Mbps. These values corresponds to the achievable throughput values when 5, 10 and 20 MB are downloaded.

5.3 Discussion

The *Throughput Estimator* (Algorithm 2), assumes that the round trip time (RTT) between the two communication end points to be constant. We have observed an instance in which there are substantial changes in the RTT during the data transfer. Hence, even though there was no drop in the congestion window growth (i.e., no loss indication in the TCP_PROBE log), the errors are high as 56%, 176%, and 289% for 5, 10, and 20 MB respectively. We removed this single sample from our evaluation. Another instance where our estimator does not perform well is where the congestion avoidance phase in TCP Cubic operates in the TCP friendly region. Our model does not take into account the TCP friendliness and in these cases, substantially underestimates the throughput. Finally, our estimator will likely not provide good estimates for very large transfers as we do not take into account packet loss. Even though losses in LTE networks have been observed to be small in our measurements with transfers of up to 20MB, as the data transfer sizes become even larger, the likelihood of more packet loss increases. The evaluation was conducted only in cellular networks. As mentioned before, usage of cellular networks are usually capped by the service providers and therefore using existing throughput estimators such as SpeedTest often consumes a large percentage of the data plan quota. Thus, the main emphasis of our throughput estimation scheme is to provide reliable estimates for throughput in cellular networks by using a small amount of data. On the other hand, WiFi networks usually do not come with data caps and therefore tests such as SpeedTest can be used.



(a) Estimated vs. Measusred Throughput (Proposed estimator)

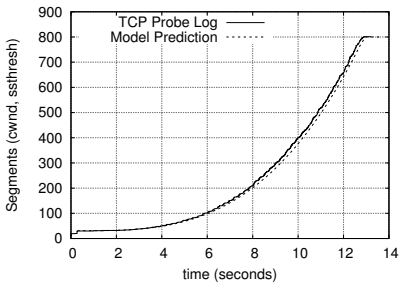


(b) Estimated vs. Measusred Throughput (Use of 1 MB throughput)

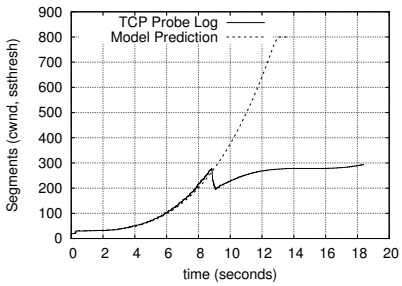
Figure 10: Scatter Plots: Estimated Throughput vs. Measured Throughput

6. CONCLUSION

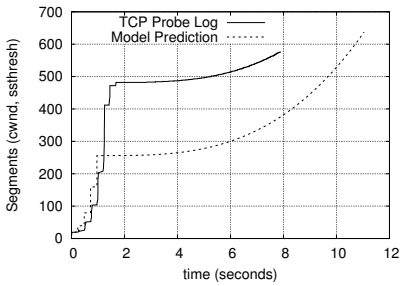
In this work, we have identified several issues related to the common throughput measurement techniques used in mobile devices. We have backed up our claim by conducting a detailed measurement study of cellular and WiFi throughput values in 10 different countries. Considering the issues in current measurement techniques, we have proposed a throughput estimator by modeling short lived TCP Cubic flows. Our proposed technique only requires minimal amount of data transfer and yet can estimate the achievable throughput with high accuracy.



(a) No fast recovery/time out event in congestion avoidance



(b) A fast recovery event triggered around the 9th second



(c) Slow start preveliant for more than 1 MB of data transfer

Figure 9: Estimated congestion window evolution vs. the logged congestion window evolution

Acknowledgments

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its International Research Centre in Singapore Funding Initiative. We would also like to thank Amazon.com, Inc. for selecting us as one of the recipients of the AWS Cloud Credits for Research Program.

7. REFERENCES

[1] comScore Inc. Major mobile milestones in may: Apps now drive half of all time spent on digital. http://goo.gl/oKHY3L. Accessed: 2016-07-06.

[2] Google AdWords blog. Inside AdWords: Building for the next moment. https://goo.gl/D6OBKJ. Accessed: 2016-07-06.

[3] 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. ACM MobiSys '12.

[4] ABI Research. Average Size of Mobile Games for iOS increased by a whopping 42% between march and september. https://goo.gl/1wDLRH. Accessed: 2016-06-14.

[5] Computerworld. Instagram, Vine short videos causing explosion on wireless networks. http://goo.gl/I9wpIj. Accessed: 2016-06-14.

[6] S. Ha, I. Rhee, and L. Xu. CUBIC: A new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating Systems Review*, 2008.

[7] W3Cook. OS Usage Trends and Market Share. http://goo.gl/Qk22r9. Accessed: 2016-07-06.

[8] Y. Xu, Z. Wang, W.K. Leong, and B. Leong. An end-to-end measurement study of modern cellular data networks. Springer PMC '14.

[9] W.L. Tan, F. Lam, and W.C. Lau. An empirical study on 3G network capacity and performance. IEEE INFOCOM '07.

[10] J. Sommers and P. Barford. Cell vs. WiFi: On the performance of metro area mobile connections. ACM IMC '12.

[11] X. Liu, A. Sridharan, S. Machiraju, M. Seshadri, and H. Zang. Experiences in a 3G network: Interplay between the wireless channel and applications. ACM MobiCom '08.

[12] Y. Chen, Y. Lim, R.J. Gibbens, E.M. Nahum, R. Khalili, and D. Towsley. A measurement-based study of multipath TCP performance over wireless networks. ACM IMC '13.

[13] S. Deng, R. Netravali, A. Sivaraman, and H. Balakrishnan. WiFi, LTE, or both?: Measuring multi-homed wireless internet performance. ACM IMC '14.

[14] N. Baranasuriya, V. Navda, V.N. Padmanabhan, and S. Gilbert. Qprobe: Locating the bottleneck in cellular communication. ACM CoNEXT '15.

[15] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. *ACM SIGCOMM Computer Communication Review*, 1998.

[16] W. Bao, V.W.S Wong, and V. Leung. A model for steady state throughput of TCP CUBIC. IEEE GLOBECOM '10.

[17] S. Poojary and V. Sharma. Analytical model for congestion control and throughput with TCP CUBIC connections. IEEE GLOBECOM '11.

[18] M. Mellia, H. Zhang, and I. Stoica. TCP model for short lived flows. *IEEE Communications Letters*, 2002.

[19] D. Zheng, G.Y. Lazarou, and R. Hu. A stochastic model for short-lived TCP flows. IEEE ICC '03.

[20] K. Zhou, K.L. Yeung, and V.O.K. Li. Throughput modeling of TCP with slow-start and fast recovery. IEEE GLOBECOM '05.

[21] S. Ha and I. Rhee. Taming the elephants: New TCP slow start. *Elsevier Computer Networks*, 2011.

[22] Linus Torvalds. linux/tcp_cubic.c at master. torvalds/linux. github. https://goo.gl/35oPwD. Accessed: 2016-05-07.

[23] H. Jiang, Y. Wang, K. Lee, and I. Rhee. Tackling bufferbloat in 3G/4G networks. ACM IMC '12.

[24] Linus Torvalds. linux/tcp_probe.c at master. torvalds/linux. github. https://goo.gl/LZG73v. Accessed: 2016-05-07.