and the core network (CN). UEs are essentially mobile handsets carried by end users. The RAN allows connectivity between a UE and the CN. It consists of multiple base stations called Evolved Node B (eNB). The centralized CN is the backbone of the cellular network. It connects to the Internet. In Figure 1, within the CN, "Monitor" is our data collection point. "SGW" and "PGW" refer to the serving gateway and the packet data network gateway, respectively. "PEP" corresponds to the performance enhancing proxy to be described shortly. From the perspective of UEs, we define *downlink* as the network path from the Internet to UEs, and *uplink* as the path in the reverse direction. Similarly, we also use the terms *downstream* and *upstream* from the perspective of the Monitor to indicate the relative locations of network elements, *e.g., downstream* refers to the path between monitor and UEs.

The Performance Enhancing Proxy (PEP). The data collection point is located within the core network of the studied LTE network. TCP traffic from or to server port 80 or 8080 traverses the PEP on the upstream side of the monitor. The PEP splits the end-to-end TCP connection into two, one between the UE and the PEP and the other between the PEP and the server. It can potentially improve the Web performance by, for example, performing compression and caching. Also the PEP makes the split transparent to UEs by spoofing its IP address to be the server's IP address.

Data Collection. Our measurement data is a large packet header trace covering a fixed set of 22 eNBs at a large metropolitan area in the U.S. The data collection was started on October 12 2012 and lasted for 240 hours. We record IP and transport-layer headers, as well as a 64-bit timestamp for each packet. No payload data is captured except for HTTP headers, the dominant application-layer protocol for today's smartphones [34]. No user, protocol, or flow-based sampling is performed. No personally identifiable information was gathered or used in conducting this study. To the extent, any data was used, it was anonymous and aggregated data. During the 10 days, we obtained 3.8 billion packets, corresponding to 2.9 TB of LTE traffic (324 GB of packet header data, including HTTP headers). To our knowledge, this is the first large real-world LTE packet trace studied in the research community.

Subscriber Identification. Due to concerns of user privacy, we do not collect any subscriber ID or phone numbers. We instead use private IP addresses (anonymized using a consistent hash function) as approximated subscriber IDs, since private IPs of the carrier are very stable. They change only at the interval of several hours. In contrast, public IP addresses observed by servers may change rapidly [4]. Private IPs can also be reused. We take this into account by using a timing gap threshold of one hour in our analysis. If a private IP has not been seen for one hour, we assume its corresponding user session has terminated. This potentially overestimates the user base, but its impact on our subsequent analyses is expected to be small since changing this threshold to 30 minutes or 2 hours does not qualitatively affect the measurement results in §4, §6, and §7. In total, we observe about 379K anonymized client IPs and 719K server IPs.

Flow Extraction. From the data set, we extract flows based on a 5-tuple of src/dst IP, src/dst port numbers, and protocol (TCP or UDP). We conservatively use a threshold of 1 hour to determine that a flow has terminated if no flow termination packets are observed. We find that similar to the idle period threshold for subscriber identification, the impact of this value on subsequent analysis results is negligible. Overall, 47.1 million flows are extracted from the trace.

We emphasize here that no customer private information is used in our analysis and all customer identities are anonymized before any analysis is conducted. Similarly, to adhere to the confidentiality under which we had access to the data, in subsequent sections, we present normalized views of our results while retaining the scientifically relevant bits.

3.2 Controlled Local Experiments

We also set up a measurement testbed in our lab for controlled experiments. The UE used is a fairly new smartphone model — Samsung Galaxy S III (SGH-I747) running Android 4.0.4 (Ice-Cream Sandwich, Linux kernel version 3.0.8) connecting to an LTE network. We tested on two large commercial LTE carriers referred to as Carrier A and Carrier B, using two Samsung Galaxy S III phones purchased from the two carriers. We configure a server with 2GB memory and 2.40GHz Intel Core 2 CPU, running Ubuntu 12.04 with `3.2.0-36-generic` Linux kernel. Both the UE and the server use TCP CUBIC as their TCP implementation.

Note that the purpose of using local experiments from a potentially different LTE carrier at locations that may not match where our studied data set comes from is to provide a different perspective and also evaluate whether observations from analyzing the data set can be empirically observed.

When measuring TCP throughput and RTT (Figures 11,20, and 19), the UE establishes a TCP connection to the server, which then transfers randomized data without any interruption. For throughput measurement, we ignore the first 10 seconds of the TCP connection (skip the slow start phase), and calculate the throughput every 500 ms from the continuously transferred data. The RTT is measured by computing the gap between timestamps of transmitting a data packet and receiving the corresponding ACK from the sender-side trace collected by the `tcpdump` tool.

4. LTE NETWORKS CHARACTERISTICS

We study LTE traffic characteristics using the aforementioned packet traces collected from the studied commercial LTE network. We also compare our results with two previous measurement studies of cellular and WiFi performance on mobile devices (§4.4).

4.1 Flow Size, Duration, Rate, Concurrency

We begin by showing the protocol breakdown of the data set. For the transport-layer protocol, TCP dominates the data set (95.3% flow-wise and 97.2% byte-wise), with majority of the remaining traffic in UDP. Within TCP, as the dominant application-layer protocol, HTTP (port 80/8080) contributes 76.6% and 50.1% of all TCP bytes and TCP flows, respectively. We also notice the popularity of HTTPS (port 443), which account for 14.8% and 42.1% of TCP bytes and flows, respectively. We present a more detailed app-layer content analysis and compare the findings with those for 3G networks in §7.1.

Following previous measurement studies of wired and WiFi networks [36, 22, 6], we are interested in three characteristics of LTE TCP flows: size, duration, and rate. Size is the total number of payload bytes within the flow (excluding IP/transport layer headers). Duration is the time span between the first and last packet of a flow. Flow rate is calculated by dividing flow size by flow duration. Understanding these characteristics is vital to many aspects in cellular networks such as eNB scheduling, usage-based billing policy, and RAN resource balancing and optimization. Our focus is TCP since its accounts for the vast majority of the traffic (95.3% of flows and 97.2% of bytes).

TCP Flow Size. Figure 2 plots the CDF of uplink and downlink payload sizes, both exhibiting strong heavy-tail distributions. Most flows are small: 90% of flows have less than 2.9 KB uplink payload and 90% of flows carry no more than 35.9 KB downlink payload. In particular, 11.3% (10.9%) of flows do not have any downlink

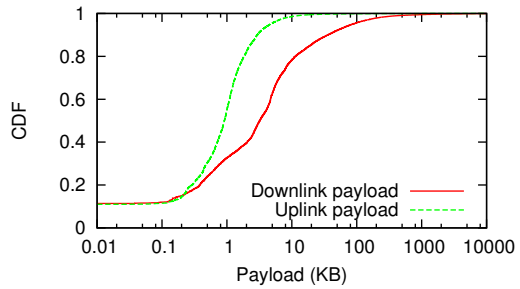

Figure 2: Distribution of TCP flow sizes.

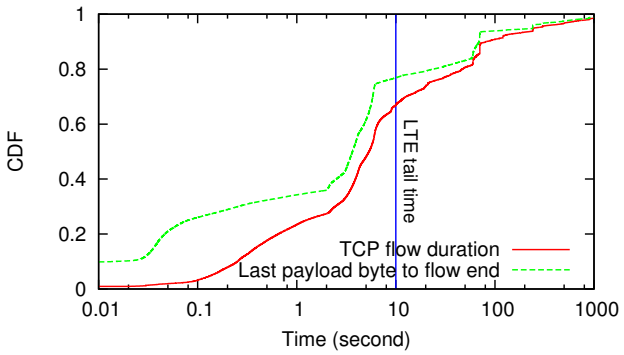

Figure 3: Distribution of flow duration and the duration between the last payload byte to the end of the flow.

(uplink) payload as they only contain complete or incomplete TCP handshakes. On the other hand, a very small fraction of large flows, which are known as "heavy-hitter" flows [22], contribute to the majority of the traffic volume. For downlink, the top 0.6% of flows ranked by payload sizes, each with over 1 MB of downlink payload, account for 61.7% of the total downlink bytes. For uplink, the top 0.1% of flows, each with over 100 KB of uplink payload, consist of 63.9% of the overall uplink bytes. Such a distribution is as skewed as that in wired networks [22].

We next examined the top 5% of downlink flows ranked by their downlink payload sizes. Each of them contains at least 85.9KB of downlink payload data and 80.3% of them use HTTP. By examining the HTTP headers (if exist) of the top 5% downlink flows, we found that 74.4% of their contents (in bytes) are video or audio. Regarding to the top 5% uplink flows, 73.6% of their bytes are images. Most of such traffic corresponds to users uploading photos to social networks such as Instagram.

TCP Flow Duration. Figure 3 shows the distribution of TCP flow duration (the solid line), defined to be the time span between the first and the last packets of a flow. Most flows are short: 48.1% are less than 5 seconds. 8.5% of the TCP flows are not even established successfully and they only consist of SYN packets. For the long-tailed part, 6.8% of the flows last at least 3 minutes and 2.8% are longer than 10 minutes.

The dotted curve in Figure 3 denotes the timing gap between the packet carrying the last payload byte and the last packet of a flow. Note that most flows in the data set are properly terminated by either FIN (86.2% of flows) or RESET (5.4%), and the remaining flows consist of only one or more SYN packets (8.5%). One example of the cause of the aforementioned timing gap is persistent HTTP that tries to reuse the same TCP connection for transferring multiple web objects so there is a timeout before the connection is closed. This does not cause any issue in wired or WiFi networks. However, in LTE networks, there exists a timeout for shutting down

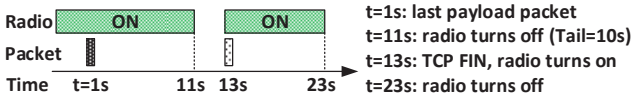

Figure 4: An example of delayed FIN packet and its impact on radio resource management.

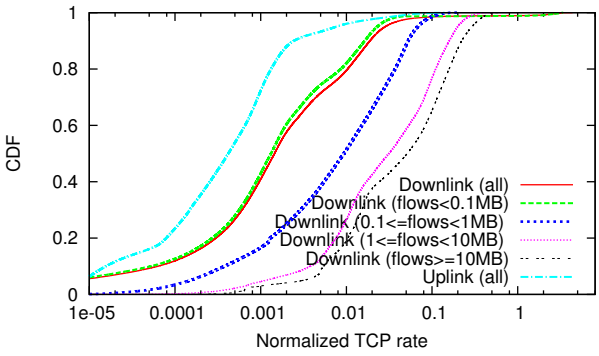

Figure 5: Distributions of normalized TCP flow rates.

the radio interface after a data transfer. Such a timeout, which is called *tail time*, saves energy by taking the device to the idle state once it finishes, and prevents frequent radio state switches [13]. We measured the timeout (*i.e.,* the tail time) to be 10 seconds for the studied LTE network. A delayed FIN or RESET packet will incur additional radio-on time and one additional off-on switch if the delay is longer than 10 seconds, leading to waste of device energy [26]. Figure 4 shows one such example, which is found to be prevalent: delaying FIN or RESET for longer than 10 seconds occurs in 23.1% of the flows in our data set as shown in Figure 3.

TCP Flow Rate. Figure 5 measures the flow rate. We observe a huge disparity between uplink and downlink rates, due to *(i)* mobile devices usually do not perform bulk data uploading (*e.g.,* FTP and P2P upload), and *(ii)* cellular uplink channel is significantly slower than the downlink channel, even in LTE networks [29]. The four downlink throughput distributions for flows with different sizes in Figure 5 indicate that larger flows tend to be faster. Previous measurements for wired networks also suggest that for Internet flows, there exist correlations among their size, duration, and rate [36, 22]. We quantitatively confirm that similar behaviors also hold for LTE flows. Let S, D, and R be downlink flow size, duration, and rate, respectively, and (X, Y) be the correlation coefficient between X and Y. We calculate the values of $(logS, logD)$, $(logD, logR)$, and $(logR, logS)$ to be 0.196, -0.885, and 0.392, respectively. For uplink flows, the values of $(logS, logD)$, $(logD, logR)$, and $(logR, logS)$ are 0.030, -0.986, and 0.445, respectively. We found the flow duration and the rate are much more negatively correlated, compared with Internet flows studied in [22], whose correlation coefficients are between -0.60 and -0.69 for Internet backbone, VPN, and DSL flows. This is worth further investigation to confirm if the sessions are terminated early due to bad performance.

Concurrent TCP Flows. We explore the concurrency of TCP flows per user in the LTE data set, as shown in Figure 6. Specifically, we use 1 second as a threshold to determine the concurrency, *i.e.,* for the sampled time point, we count the number of TCP flows that have the downlink data transfers within the last 1 second. We observe that for 72.1% of the time, there is only one TCP flow actively downloading data, and this percentage might be even larger for smartphone users, considering that our data set also consists of a small share of users that uses LTE data cards on their laptops, which may have high TCP flow concurrency.

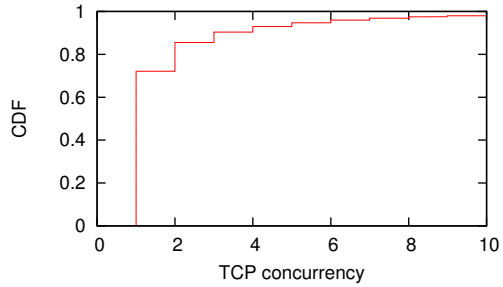

Figure 6: Concurrency for TCP flows per user uniformly sampled by time.

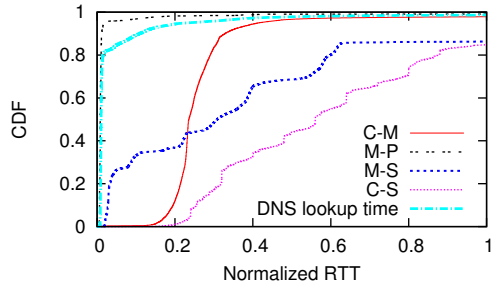

Figure 7: Distributions of normalized handshake RTT and DNS lookup time.

4.2 Network Latency

Figure 7 measures distributions of TCP handshake RTT. "C", "M", "P", and "S" correspond to the client (UE), monitor (the data collection point), PEP, and remote server, respectively. Since the monitor lies in the LTE core network, we can break down the overall RTT into two components: the *downstream* RTT between a client and the monitor ("C-M", for all traffic), and the *upstream* RTT between either the monitor and the PEP ("M-P", for TCP port 80/8080 traffic) or server ("M-S", for other traffic). The downstream RTT is an estimation of the latency in the RAN (Figure 1). In a TCP three-way handshake, let the monitor's reception time of SYN (uplink), SYNACK (downlink), and ACK (uplink) be t_1, t_2, and t_3, respectively. Then the upstream RTT is computed as $t_2 - t_1$, and the downstream RTT is $t_3 - t_2$. The "C-S" curve combines both the "C-M" and the "M-S" components (for non-PEP traffic only).

It is well known that in 2G/3G data networks, usually the RAN latency dominates the overall end-to-end delay [35]. This is no longer the case in LTE networks. Figure 7 shows that the upstream RTT to a remote server ("M-S") has a higher variance, and is usually larger than the downstream RTT ("C-M"). This is further confirmed by Figure 8, which plots the distribution of ratios between the upstream RTT and the downstream RTT for non-PEP ("C-S") flows. For 55% of the non-PEP flows, their upstream RTTs are larger than the corresponding downstream RTT, whose reduction (*i.e.,* the reduction of the RAN latency) is mostly attributed to the flattened network topology in the LTE RAN. For example, the two-layered RAN architecture (NodeB and the Radio Network Controller, RNC) in 3G UMTS/HSPA networks is replaced by the single-layered eNB architecture in LTE, helping significantly reducing the RAN latency [29] (See §4.4 for quantitative comparisons). Further, the "M-P" curve in Figure 7 indicates the latency between the monitor and the PEP is very small.

LTE Promotion Delay. In cellular networks, the end-to-end latency of a packet that triggers a UE's radio interface to turn on is significantly long. Such a packet incurs a radio resource control (RRC) promotion delay during which multiple control messages

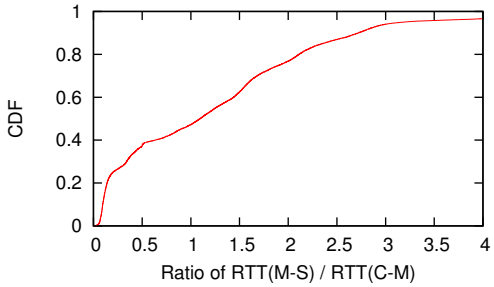

Figure 8: Distribution of the radio between uplink and downlink RTT (for non-PEP traffic).

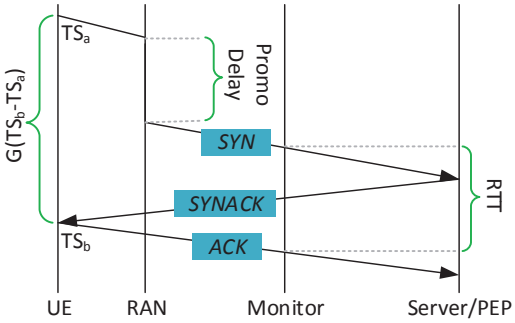

Figure 9: Estimating the promotion delay.

are exchanged between a UE and the RAN for resource allocation. The promotion delay can be as long as 2 seconds in 3G networks [25], and it also exists in LTE networks [13]. The promotion delay is not included in either the upstream RTT or the downstream RTT in Figure 7, since the promotion (if any) has already finished when the monitor observes a SYN packet, as illustrated in Figure 9. However, we are able to infer the promotion delay using the TCP timestamp embedded into a TCP packet when the packet is about to leave the UE. In a three-way handshake, let the TCP timestamp of the SYN and the ACK packet be TS_b and TS_a, respectively. Then the round-trip time (including the promotion delay) experienced by the UE is $G(TS_b - TS_a)$ where G is the inverse of the ticking frequency of UE's clock generating the TCP timestamp. Note that the TCP timestamps are not wall-clock times. Their units depend on the ticking frequency of the UE. We detail how to compute G in §6.1. Finally the promotion delay (if exists) could be derived by subtracting the RTT between the UE and the server/PEP (estimated in Figure 7) from $G(TS_b - TS_a)$, as shown in Figure 9.

We calculated promotion delays using the aforementioned method, by examining TCP handshakes with the following property: the user does not send or receive a packet within the time window $(t-T, t)$ where t is the reception time of SYN and T is the window size. We conservatively choose $T = 13$ seconds which is larger than the 10-second timeout of the studied LTE network. This restriction ensures the UE is in the idle state when the handshake is initiated. Therefore, the SYN packet must trigger a state promotion. The 25%, 50%, and 75% percentiles of the promotion delay are 319 ms, 435 ms, and 558 ms, respectively. We found these are significantly shorter than the 3G promotion delays (around 2 seconds from idle to high-power state, and around 1.5 seconds from low-power to high-power state [25]), possibly due to the simplified signaling protocol in LTE networks [29].

DNS Lookup. The "DNS" curve in Figure 7 measures the DNS lookup delay, computed as the delta between the reception time of a DNS request and its response at the monitor. Note this is the latency between monitor and the DNS server, and we are not able to measure the downstream latency since DNS messages are transferred

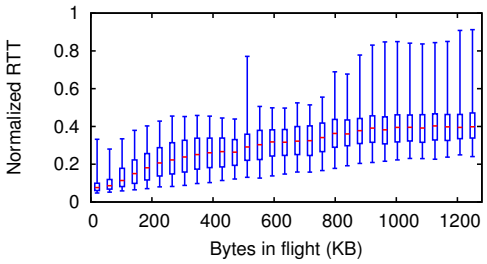

Figure 10: Downlink bytes in flight vs. downstream RTT.

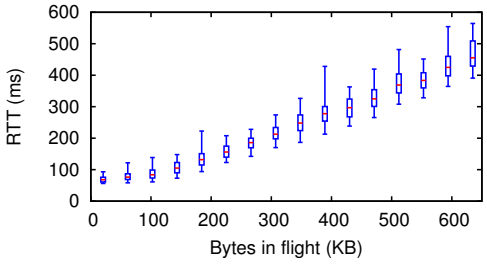

Figure 11: Downlink bytes in flight vs. downstream RTT (controlled lab experiments with LTE Carrier A).

over UDP. We found that the upstream latency is usually very short *i.e.,* less than 10 ms for 87.3% of request-response pairs. Since the studied LTE network (Figure 1) has its own DNS server, the short lookup delay indicates the desired effectiveness of the DNS server, which caches most DNS responses so their domain names are effectively resolved locally within the LTE core network.

4.3 Queuing Delay and Retransmission Rate

§4.2 focuses on the RTT of TCP connection establishment during which the small TCP handshake packets are usually unlikely to be buffered by the network. During the data transfer phase, a TCP sender will increase its congestion window, allowing the number of unacknowledged packets to grow. Such "in-flight" packets can potentially be buffered by routers and middleboxes on their network paths, incurring queueing delays. In LTE networks, buffers are extensively used to accommodate the varying cellular network conditions and to conceal packet losses [29].

Figure 10 shows the relationship between the downstream RTT and the number of downlink in-flight bytes, which is computed by counting the unacknowledged bytes. As shown in Figure 10, the downstream RTT tends to inflate as the number of in-flight bytes increases. The in-flight bytes in our studied LTE network can be larger than 1200 KB, causing high latency due to the queuing delay. We vary this in local experiments (§3.2) where we measure both the RTT and the bytes in flight at UE for two large commercial LTE networks. As shown in Figure 11 and 12, the trend that RTT grows with the number of in-flight packets is obvious. Our observation is also consistent with a recent study [16] that shows the usage of large buffers in today's cellular networks may cause high queuing delays. In addition to that, we further demonstrate its prevalence in today's LTE networks: as shown in Figure 13, which plots the distribution of downlink in-flight bytes for large flows ($>$ 1MB), about 10% of measured instances have in-flight bytes greater than 200 KB, potentially leading to long queuing delays.

Clearly, for short flows or traffic triggered by user interactions (*e.g.,* web browsing), queues are not likely to build up. For long-lived flows, usually it is the throughput instead of latency that matters. However, when short-lived and long-lived flows coexist (*e.g.,* performing browsing while streaming in the background), queuing delay may severely deteriorate user experience by introducing unacceptable delays for short flows. Moreover, as a new observation, we found that a high downstream queuing delay may often cause TCP's congestion window to collapse upon a single packet loss. We discuss this newly identified and rather severe issue in §5.

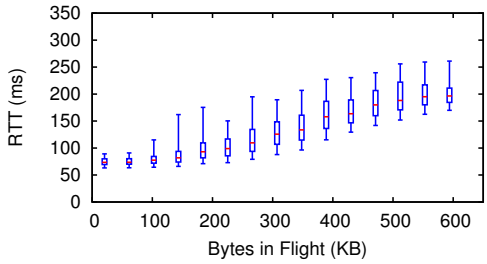

Figure 12: Downlink bytes in flight vs. downstream RTT (controlled lab experiments with LTE Carrier B).

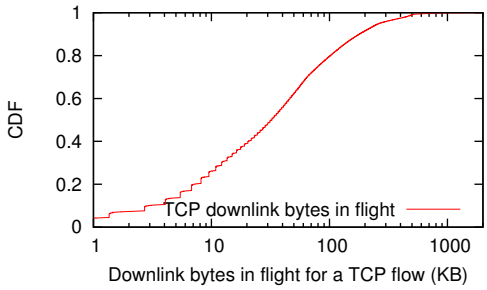

Figure 13: Distribution of downlink bytes in flight for large flows ($>$ 1 MB).

Retransmission Rate. We study TCP downlink retransmission rate, defined as the number of retransmitted packets divided by all packets, across all downlink flows in our data set. 38.1% of the flows have zero retransmission, and the median is only 0.06%. Such low retransmission rates are comparable to those in wired networks [22]. There are even fewer packet losses since the retransmission rate is an upper bound of the packet loss rate in the downstream (*i.e.,* between UE and monitor, note we are not able to capture losses occurring on the upstream side of the monitor). In fact, in cellular networks, most transport-layer losses are concealed by physical/MAC-layer retransmission and reduced by buffering. In particular, buffers in LTE networks upstream from the airmile can play an important in absorbing the burstiness of the traffic transmitted over the lossy wireless link, helping achieve a low loss rate.

4.4 Comparison to Previous Studies

We compare our results with three previous measurement studies, focusing on three important metrics: TCP downlink throughput, TCP uplink throughput, and TCP handshake RTT. The 3GTest study [14] deployed an app that measures network performance metrics on users' handsets. Their data consisted of 35K cellular (3G only) tests from customers of four large U.S. cellular carriers in late 2009. The 4GTest study [13] adopts a similar approach while focusing on LTE users. Its data comprises of about 1K LTE tests and a few WiMAX tests across the U.S. in late 2011. A recent study [31] examined a 15-week data set from `speedtest.net` in 2011. Table 1 shows their reported performance metrics for handheld device users from three locations: New York City (246K WiFi tests / 79K cellular tests), Madison Wisconsin U.S. (24K WiFi / 4K cellular), and Manchester U.K. (291K / 31K). The cellular technology ranges from 2G EDGE to 4G LTE, but is dominated by 3G (UMTS/EvDO/HSPA).

We discuss three major issues that may affect the comparison. First, all three previous studies perform throughput measurement

Table 1: Comparing with previous measurement studies

Study	Our Results	3GTest [14]	4GTest [13]		SpeedTest [31]					
Time	October 2012	Aug to Dec 2009	Oct to Dec 2011		February 21 2011 to June 5 2011 (15 weeks)					
Location	One US Metro Area	Across U.S.	Across U.S.		New York City		Madison WI, US		Manchester UK	
Type	LTE Only	Four 3G ISPs	LTE	WiMAX	Cellular	WiFi	Cellular	WiFi	Cellular	WiFi
5% TCP DL*	569	74 – 222**	2112	431	108	404	99	347	28	267
50% TCP DL	9185	556 – 970	12740	4670	1678	7040	895	5742	1077	4717
95% TCP DL	24229	1921 – 2943	30812	10344	12922	17617	3485	14173	3842	15635
5% TCP UL	38	24 – 52	387	172	52	177	55	168	25	180
50% TCP UL	2286	207 – 331	5640	1160	772	2020	478	1064	396	745
95% TCP UL	8361	434 – 664	19358	1595	5428	10094	1389	5251	1659	5589
5% HS RTT	30	125 – 182	37	89	68	21	99	24	98	34
50% HS RTT	70	160 – 200	70	125	159	54	184	69	221	92
95% HS RTT	467	645 – 809	127	213	786	336	773	343	912	313

* TCP DL: downlink throughput (kbps). TCP UL: uplink throughput (kbps). HS RTT: TCP handshake RTT (ms). 5%, 50%, 95% are percentiles.
** For a range $x - y$, x and y are the result of the worst and the best carriers, respectively, for that particular test.

using bulk data transfer of a large file without any pause while our flows may consist of idle time periods (*e.g.,* due to user think time), leading to a lower throughput. To obtain more fair comparison, here we only consider large non-PEP flows in our data set (with at least 200 KB for uplink and 1 MB for downlink) with no visible idle time period (with maximum inter-packet time of less than 1 second, which is larger than 99.9^{th} percentile of RTT). Second, in our case, remote servers may impose rate limit [10] while all previous studies perform active probing using dedicated test servers without any limitation on throughput. Third, we infer performance metrics from traces of real Internet servers, while 3GTest, 4GTest, and SpeedTest employ different server selection policies: 3GTest uses a single server located in U.S. while 4GTest and SpeedTest picks a server geographically close to the UE. This in particular affects the latency estimation.

The comparison results are shown in Table 1. Despite aforementioned differences among diverse measurement approaches, we believe the comparison can still demonstrate the advantage of LTE over other types of cellular access technology, since their performance difference is quite significant: the median downlink throughput, uplink throughput, and handshake RTT are 9.5x, 6.9x, and 0.43x compared with the median values of the best U.S. 3G carrier in 2009, respectively. Compared with the 2011 New York City cellular results, the ratios are 5.5x, 3.0x, and 0.44x for DL throughput, UL throughput, and RTT, respectively. Moreover, on mobile devices, LTE also outperforms WiFi in many cases. Specifically, for the 5^{th}/50^{th}/95^{th} percentiles of downlink throughput and the median uplink throughput shown in Table 1, LTE performs better than WiFi. Based on Table 1, LTE's latency appears higher than that of WiFi. However, recall that Speedtest always picks a nearby test server while we are measuring the RTT between UE and real servers that may be far away. This may lead to an unfair RTT comparison. Furthermore, our performance values are consistently lower than those reported by LTE tests from 4GTest, very likely due to the rate limiting imposed by remote servers as mentioned before. A recent study [10] indicates such rate limiting is prevalent across today's Internet servers. We also observe that LTE significantly outperforms WiMAX in all three metrics.

5. ABNORMAL TCP BEHAVIOR

Due to their resource usage, we focus on *large flows* defined to be relatively long flows, with more than 5 seconds data transfer time, and total downlink payload exceeding 1MB. These large flows account for only 0.3% of all TCP flows in our data set, but their total downlink payload contributes to 47.7% of all downlink payload.

As background, upon receiving an out-of-order unacknowledged segment, a TCP receiver sends an immediate duplicate ACK [3].

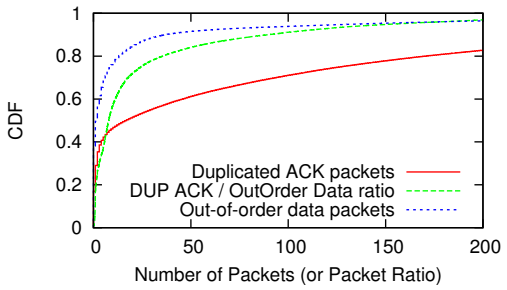

Figure 14: Observed duplicate ACKs and packet reordering in large TCP flows.

From the sender's perspective, duplicate ACKs can be caused by reordering or loss. Therefore, when there is a large amount of bytes in flight and one data segment S is lost, each data segment with sequence number higher than that of S triggers a duplicate ACK, before a retransmission of S is successfully received. So a long sequence of duplicate ACKs strongly suggests a packet loss. When TCP detects 3 duplicate ACKs, it infers a data packet loss and retransmits it according to the fast retransmit [3]. In the monitor traces, we detect this behavior as the data packet sent by fast retransmit is out-of-order relative to other packets.

Figure 14 summarizes duplicate ACKs and packet reordering in the large TCP flows. Although the median of duplicate ACKs in large flows is 17, for over 29.0% of the large flows, there are over 100 duplicate ACKs. We observe that the number of out-of-order data packets in large flows is substantially smaller than that of duplicate ACKs, with a median value of only 2. By studying the ratio between duplicate ACKs and out-of-order data packets, 24.7% of flows have a ratio of over 25, and for some flows, this ratio can reach up to 5,000. This indicates that even a single out-of-order data packet can trigger a large number of duplicate ACKs when the bytes-in-flight are large, using up more uplink bandwidth.

Fast retransmission allows TCP to directly send the lost segment to the receiver possibly preventing retransmission timeout (RTO). If so, TCP would resume data transfer with the congestion window size reduced by half. However, as shown earlier, we identified significant queuing build up between UE and monitor. Such large in-network queues capable of holding up to a few megabytes data could delay the receipt of the retransmitted data packet. In that case, if TCP does not use duplicate ACKs to update RTO (retransmission timeout), a timeout is likely to happen. Specifically, if the corresponding ACK does not arrive at the server within the (underestimated) RTO, the congestion window would drop to 1 segment, triggering slow start, significantly hurting TCP performance. We refer to this as the *undesired slow start* problem.

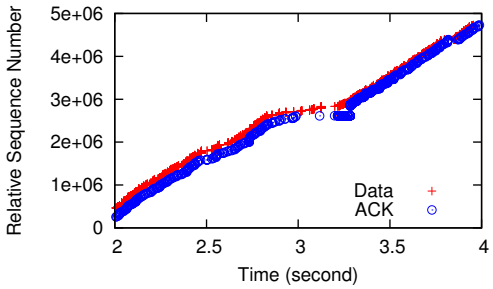

Figure 15: Duplicate ACKs not triggering a slow start.

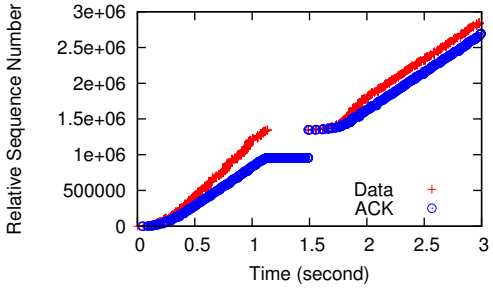

Figure 16: Duplicate ACKs triggering a slow start.

Figures 15 and 16 demonstrate two examples in the data set, where Figure 15 shows that the train of duplicate ACKs does not trigger slow start and Figure 16 includes a case where an undesired slow start is triggered. Their key difference is that Figure 16 has about 500KB bytes in flight before the first duplicate ACK, while Figure 15 has much fewer bytes in flight.

In TCP, RTO is computed by the sender using smoothed RTT and RTT variation [20]. However, using duplicate ACKs to update RTO, which may be beneficial by allowing more accurate RTT estimation, is not standardized. In Figure 16, between 1.1s and 1.5s, the sender receives many duplicate ACKs. Due to the growing queueing size, RTT grows from 262ms (the last RTT sample before the first duplicate ACK) to 356ms, the RTT for the retransmitted packet. The sender's TCP implementation apparently ignores these duplicate ACKs for updating RTO, which remains the same without the duplicate ACKs being considered. Following the method for calculating RTO [20], we observe that RTO is around 290ms before the first duplicate ACK, which is smaller than the RTT of the retransmitted packet (356ms). This problem does not happen in Figure 15, because the RTT before the first duplicate ACK is close to that after the last duplicate ACK, due to the small number of bytes in flight. Although it is recommended that the RTO should be at least 1 second [20], depending on the operating systems, different minimum values are used, *e.g.,* Linux's minimum RTO is 200ms [28]. Such small values of RTO can exacerbate the undesired slow start problem demonstrated in Figure 16.

We study the prevalence of the undesired slow start problem. To tell whether there is a slow start following a long list of duplicate ACKs, we use a heuristic metric R_{ss}, the ratio of slow start: $R_{ss} = \frac{\theta_{[100,200]}}{\theta_{[0,100]}}$, where $\theta_{[t_1,t_2]}$ is the average downlink throughput from t_1 ms to t_2 ms after the last duplicate ACK. We empirically choose 200ms as it is observed to be shorter than a typical slow start in the LTE networks. During a slow start, R_{ss} is expected to be larger than that when there is no slow start. For example, the R_{ss} is 1.0 for Figure 15 and R_{ss} is 3.7 for Figure 16. In practice, we observe that 1.5 is a good threshold for R_{ss} in determining slow start. Using this threshold, we have determined that for all the large TCP flows with at least one lost data packet, 20.1% of them

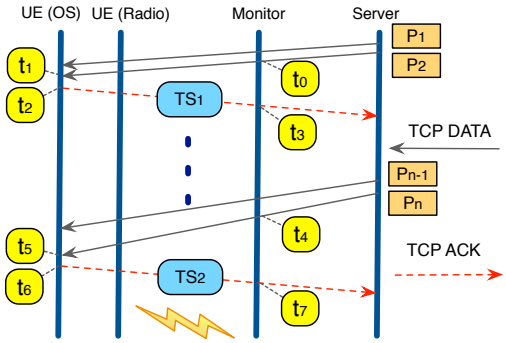

Figure 17: Typical TCP data transfer.

suffer from the slow start problem, which consists of 12.3% of all large TCP flows. In one case, a 153-second flow even experience 50 slow starts, resulting in an average throughput of only 2.8Mbps, while the estimated bandwidth actually larger than 10Mbps.

There are different ways to mitigate this problem. One approach is to update RTO with the help of duplicate ACKs with TCP Selective Acknowledgment options (SACK) [18]. By taking the difference between the SACK window of two consecutive duplicate ACKs, we can usually identify the exact data packets corresponding to these ACKs. If there exists ambiguity (*e.g.,* due to lost or reordered ACK packets), we can simply ignore the corresponding samples. In our data sets, packet reordering rate is less than 1% and SACK is enabled in 82.3% of all duplicate ACKs, making this approach promising. If SACK is disabled, we can use a fall-back approach to estimate RTT based on duplicate ACKs by assuming that they are in response to the data packets sent out in order. This assumption holds in most cases as the packet reordering rate is low.

Using the above approaches, we can obtain RTT estimations for duplicate ACKs and update RTO accordingly, which effectively prevents the timeout of retransmitted packets due to increased queueing delay. Our initial analysis shows that these two approaches can prevent more than 95% of the undesired slow starts. Note that the RTT estimation method used in TCP Vegas [5] with help of the TCP timestamps option is not applicable to duplicate ACKs, since the echo timestamps of all duplicate ACKs are all the same, with their values being the timestamp of the segment before the lost segment, rather than the timestamp of the data segments triggering the duplicate ACKs. From the mobile network operators' perspective, one solution of the undesired slow start problem might be to prioritize retransmitted packets. However, the security and performance implications of this approach are yet to be studied.

6. BANDWIDTH ESTIMATION

In order to understand the network utilization efficiency of existing applications in the LTE networks, we first need to know the available bandwidth for each user. Previous work on *active* bandwidth measurement methodology to estimate available bandwidth, *e.g.,* using packet pairs, packet trains, and parallel TCP connections [21, 15, 12], do not apply here. As existing studies have shown that network condition is highly variable in cellular networks [14], active probing would require us to launch measurements for each user at the time of trace collection. Using packet traces collected at the monitor, we instead devise a *passive* bandwidth estimation algorithm to capture the available bandwidth for each user using TCP flows that may not fully utilize the bandwidth.

6.1 Bandwidth Estimation Algorithm

Figure 17 illustrates a typical TCP data transfer. Our monitor lies between the server and the UE, and we only use packet traces

collected at the monitor for analysis. The high-level idea for our bandwidth estimation algorithm is to select a time window within which the sending rate is fast enough to exceed the available bandwidth, and then calculate the UE receiving rate that corresponds to the actual available bandwidth during the short time window. Note that the receiving rate is often smaller than the sending rate, causing the in-network buffers to be filled up (§4.3).

We use Figure 17 to illustrate our bandwidth estimation algorithm. At t_2, UE sends an ACK in response to the two data packets P_1 and P_2. And similarly, at t_6, the ACK for P_{n-1} and P_n is sent. From the monitor's traces, we observe that $n-2$ data packets ($P_3 \cdots P_n$) are sent to the UE in a time window between t_0 and t_4. Assuming the average payload size of these $n-2$ packets is S, the sending rate between t_0 and t_4 is

$$R_{snd} = \frac{S(n-2)}{t_4 - t_0} \quad (1)$$

At UE, the receiving rate for these $n-2$ packets is

$$R_{rcv} = \frac{S(n-2)}{t_5 - t_1}$$

Typically, t_2 is very close to t_1 and similarly $t_5 \approx t_6$. In our controlled lab experiments, for a 30-minute continuous trace, the median value of the delay between a data packet and the corresponding ACK is negligible: 0.3ms. However, such a delay, *e.g.,* $t_2 - t_1$, could be large in some cases. Typically, one ACK in TCP is for two data packets and when there is only one data packet pending acknowledgement, the receiver may delay sending the ACK by up to 500 ms, which is known as the delayed acknowledgement mechanism [27]. In our example, if P_{n-1} has already been acknowledged by another ACK and after t_5 there is no more data packet arriving at the UE side, the ACK for P_n could be delayed. For simplicity, we ignore cases where the last ACK is acknowledging only one data packet, indicating it might be a delayed ACK. We also do not consider cases with out-of-order data packets or duplicate ACKs in the time window for bandwidth estimation, as there may be ambiguity in packet timing. Then we have

$$R_{rcv} \approx \frac{S(n-2)}{t_6 - t_2}$$

If the uplink delay from the UE to the monitor is stable, we can assume $t_6 - t_2 = t_7 - t_3$. However, this assumption may not hold as RTT can be significantly affected by the bytes in flight. Instead, we use the TCP Timestamps option [32] to calculate $t_6 - t_2$. If that option is enabled, ACKs sent from a UE will contain the Timestamp Value field (*TSval*) *i.e.,* the current value of the UE's timestamp clock. The unit for *TSval* depends on devices' implementation. We denote it by G, which can be treated as a constant for the same device. Assuming G is known, we can estimate R_{rcv} as

$$R_{rcv} \approx \frac{S(n-2)}{G(TS_2 - TS_1)} \quad (2)$$

where TS_1, TS_2 are the *TSval* in the two corresponding ACKs. Our bandwidth estimation algorithm only requires a UE having the TCP Timestamps option enabled. In our data set, for 92.6% of the TCP flows, this requirement is satisfied.

We infer G using the method from previous work [11]. Using the example in Figure 17, we have

$$G \approx \frac{TS_2 - TS_1}{t_7 - t_3} \quad (3)$$

To minimize the error, we require $t_7 - t_3$ to be large enough *i.e.,* greater than a threshold δ_G. A larger δ_G value leads to more accurate estimation of G but requires more time for inference. Figure 18 plots the relationship between δ_G and the estimation error rate for G, based on controlled experiments for two devices, whose actual G values (*i.e.,* the ground truth) are measured at the UE side using 30-minute long traces. We observe that the error rate of G inference drops significantly as δ_G increases so we conservatively select $\delta_G = 3$ seconds, which incurs less than 0.1% of error rate in Figure 18. The error rate also depends on the value of G. In our data set, among all large flows, 5.9% of them do not have the UE Timestamps option enabled, 57.3% have $G \approx$ 1ms/tick, 36.4% have $G \approx$ 10ms/tick, and the rest 0.4% have other values, *e.g.,* $G \approx$ 100ms/tick. With $\delta_G = 3$ seconds, the error rates of inferred G are less than 0.1% for the vast majority of large flows.

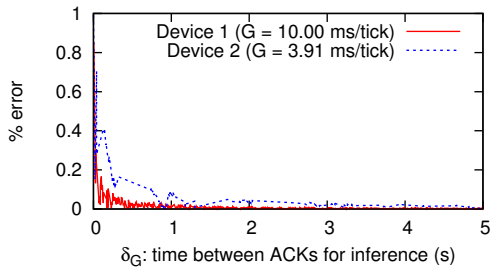

Figure 18: G inference and the selection of δ_G.

Summary. for a target TCP flow, if its G value is not known, the algorithm uses the initial $\delta_G = 3$ seconds of the flow to infer G by selecting two uplink packets that are at least δ_G seconds apart (Formula 3). Flows without UE TCP Timestamps are ignored. Then the algorithm scans for each time window with high sending rate R_{snd} calculated by Formula 1. If *(i)* $R_{snd} \geq C$, a pre-known constant of the maximum possible available bandwidth in the studied network, and *(ii)* there is no out-of-order data packets or duplicate ACKs within the time window, and *(iii)* the last ACK in the window is not a delayed ACK, then the algorithm computes a bandwidth estimation sample according to Formula 2. The selection of C incurs the following tradeoff: if C is too small, the bandwidth will be underestimated when the sending rate within the estimation window is not high enough to fully utilize the available bandwidth; if C is too large, we may not be able obtain a sufficiently large number of estimation samples. We conservatively choose C = 30Mbps, which is verified to be higher than the rate of most flows, and in the meanwhile allows us to predict the available bandwidth for over 90% of the large downlink flows. Our algorithm automatically searches for different window sizes $t_5 - t_1$ for getting bandwidth estimation samples, and we only consider the cases where there are packets at both t_1 and t_5. Typically, valid bandwidth samples are obtained when the window size is equal to a few times the RTT.

In addition to downlink bandwidth, our algorithm is also applicable to uplink bandwidth estimation, by interchanging the UE and the server in Figure 17. Similarly, our bandwidth estimation algorithm also works in other network types, such as 3G, WiFi and even wired networks, with proper parameter settings of C and δ_G.

Although the described algorithm is based on one single TCP flow per user, a similar idea can be applied to multiple concurrent flows per user by summing up the predicted bandwidth for different flows. As long as we ensure that the total sending rate for all concurrent flows are larger than C, the aggregated receiving rate would be an accurate estimation of the available bandwidth. In this study, we apply the algorithm on LTE downlink traffic (UEs downloading contents from servers) for single TCP flows, *i.e.,* without other competing downlink flows for the same user.

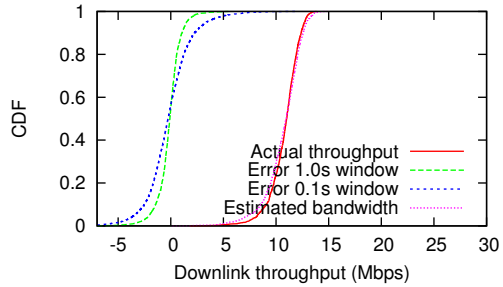

Figure 19: CDF of bandwidth estimation results for LTE network (controlled lab experiments with Carrier A).

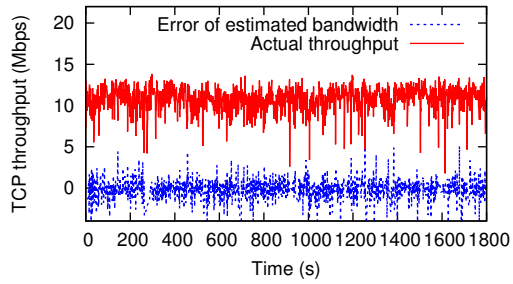

Figure 20: Time series of bandwidth estimation for LTE network (controlled lab experiments with Carrier A).

6.2 Validation with Local Experiments

To validate the bandwidth estimation algorithm, we use controlled experiments with their setup described in §3.2.

Recall that during the throughput test (§3.2), the server sends data without any interruption so the throughput measured on the UE is a reasonable (but not necessarily perfect) approximation of the available bandwidth. Therefore in Figure 19, we compare the distribution of estimated bandwidth calculated from the server-side packet trace (§6.1) with the actual throughput measured from the UE-side packet trace by sliding a window of a fixed length (*e.g.,* 1.0s) over the trace. For each window position, we get one server-side bandwidth estimation sample that is time-wise closest to the center of that window, and we compare this sample with the actual throughput to obtain an error sample. Note that the term "error" here is relative to the actual throughput observed from UE-side traces, which itself might not be the actual available bandwidth, and the true error rate for our estimation algorithm could be even smaller. The error distributions for two window lengths, *i.e.,* 1.0s and 0.1s, are shown in Figure 19. For the 1.0-second window, the average error is 7.9% and for 0.1s window, the UE throughput has higher variation and the average error is slightly higher. Figure 19 also directly compares the distributions of the absolute values of the actual throughput (using a sliding window length of 1.0s) and the estimated bandwidth, both of which are very close. Figure 20 visualizes an example test by showing the UE-perceived throughput as well as the absolute error for the estimated bandwidth over 30 minutes (1.0s window). The actual throughput fluctuates around 10Mbps and the error fluctuates within $\pm$1Mbps in most cases.

6.3 Bandwidth Utilization by TCP Flows

In this section, we analyze the LTE traffic data set to understand network utilization efficiency of TCP flows. As shown in Figure 6 (§4.1), most users have only one TCP flow actively downloading data. We therefore only consider single TCP flows with no competing downlink flows from the same user.

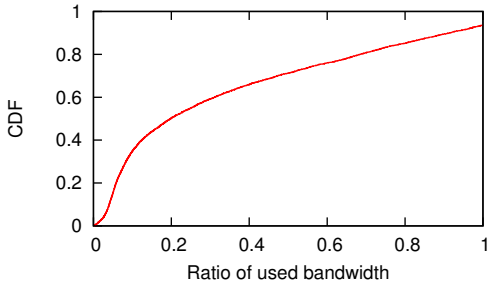

Figure 21: BW utilization ratio for large downlink TCP flows.

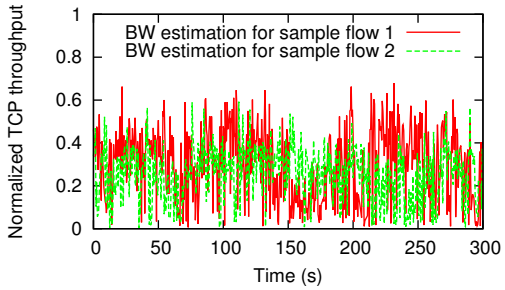

Figure 22: BW estimation timeline for two large TCP flows.

We apply the bandwidth estimation algorithm on the large TCP downlink flows (>5 seconds, >1 MB) that are not concurrent with other flows. We split each large flow into consecutive windows of 250ms. For each window, we take one bandwidth estimation sample that is closest to the center of the window. For some flows, there exist windows that do not contain any valid bandwidth estimation sample and we simply ignore such windows. This will not qualitatively affect the analysis results as such unknown duration accounts for less than 20% of the total flow duration. For each flow, we use the average value of all bandwidth estimation samples as the estimated flow bandwidth and compare it with the actual utilized flow bandwidth, computed by dividing total bytes by flow duration.

Figure 21 plots the ratio of used bandwidth to estimated bandwidth across large flows. The median ratio is only 19.8%. For 71.3% of large flows, their bandwidth utilization ratios are below 50%. For 6.4% of the flows, the used bandwidth is slightly larger than the estimated bandwidth, possibly due to estimation error. On average, the utilization ratio is 34.6%. Transferring the same amount of data requires a longer period of time with lower bandwidth utilization ratio, which incurs additional radio energy overhead and more radio resource consumption [13].

Figure 22 shows two sample large TCP flows and their estimated bandwidth in the LTE data set. They belong to two users at different time and the time is aligned only for presentation purpose. We observe that the available bandwidth varies significantly over time and even on the scale of seconds. This could be attributed to network condition changes (*e.g.,* signal strength) or changes of the network load in associated eNB. In order to dissect the root cause of such variability, more information, *e.g.,* load information of eNB, is needed.

To understand how well TCP performs under highly variable available bandwidth, we use `iptables` to redirect packets to a packet scheduler we designed, which changes the available bandwidth following the variations observed in LTE networks. The packet scheduler also injects varying delays to each packet helping us to understand the impact of RTT. Intuitively, when RTT is larger, TCP would adapt slower to the varying available bandwidth, as the congestion window is updated only once per RTT. We measure the

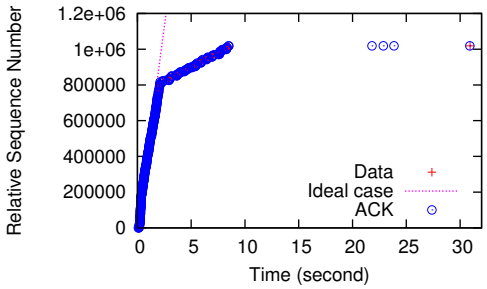

Figure 23: Full receive window slows Shazam player (a popular app) in downloading a 30-second music file.

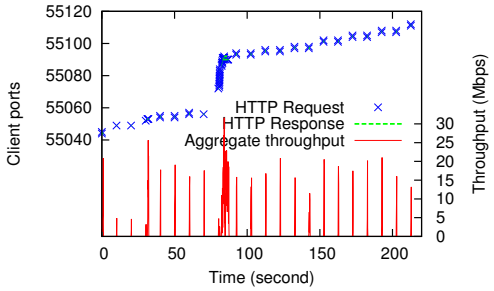

Figure 24: The periodic request behavior of Netflix player limiting its overall throughput.

bandwidth utilization ratio with the packet scheduler changing the available bandwidth every 500ms. We observe that under small RTTs, TCP can utilize over 95% of the available bandwidth. However, when RTT exceeds 400ms, the utilization ratio drops to below 50%. We also observe that for the same RTT, higher bandwidth variation leads to lower bandwidth utilization. These observations further suggest that large RTTs can degrade TCP performance in LTE networks, which have inherently varying available bandwidth likely caused by changes in load and channel conditions.

7. NETWORK APPLICATIONS IN LTE

In this section, we characterize the network applications and traffic patterns in the LTE data set. Specifically, we study the application-layer causes of inefficient bandwidth usage observed in §6.

7.1 HTTP Content Characterization

HTTP dominates the application-layer protocol usage on mobile devices [34]. We break down the total bytes of HTTP traffic based on content types. About 37.8% are video, followed by 19.5% for images and 11.8% for text. Files in `zip` format contribute to 8.3% of the HTTP traffic and they mostly correspond to file downloads such as app updates. Audio contents are responsible for 6.5% of HTTP traffic and other content types for 5.6%. The remaining 10.5% are unknown. Within video contents, we observe 12.9% to be *octet-stream* type (byte stream in binary format), most of which are generated by video players via byte-range requests.

Previous studies show that the multimedia contents (video and audio) correspond to 40% of the traffic generated by mobile handheld devices in DSL networks [19], and video contributes to 30% of the 3G cellular traffic [7]. Although we observe slightly higher percentage of multimedia traffic in this LTE network, the difference is insignificant. Overall, we observe multimedia contents still dominate the LTE traffic, followed by images.

7.2 Inefficient Network Usage

We investigate the large flows with under-utilized network bandwidth and observe that the TCP receive window size [32] has become the bottleneck in many cases.

Figure 23 shows one such example: an iOS user launches the popular Shazam app [2] to download a 30-second music file of 1MB. Initially, the data transfer speed is high and between time 0s and 2s the average downlink throughput is over 3Mbps. However, between 2s and 9s, the average throughput decreases to less than 300Kbps. The total download time is 9 seconds and as indicated by the *ideal case* curve, the download could have been completed within 2.5s, based on our estimation of the available bandwidth. In addition, we notice that the TCP connection is not immediately closed after the file transfer is complete, although the HTTP request specifies the connection to be `Connection:close`. In fact, the connection is torn down at 30s, after the music clip has finished playing and the client sends some TCP receive window updates to the server between 20s and 25s to increase receive window sizes. Overall, the total download process keeps the radio interface on for 38 seconds. Assuming a tail time of 10 seconds [13], in the ideal case, the active radio time is only 12.5 seconds.

The performance drop at 2s in Figure 23 is due to the TCP receive window becoming full. Between 0s and 2s, the window size has gradually dropped to a small value, *e.g.,* at the turning point around 2s, the window size is 816 bytes, even smaller than the maximum packet payload size (1358 bytes in the trace). As TCP rate is jointly controlled by the congestion window and the receive window, a full receive window would prevent the server from sending more data regardless of the congestion window size, leaving the bandwidth under-utilized.

The reason for the full TCP receive window is two-fold. First, the initial receive window size is not large, *e.g.,* 131.8KB in the case of Figure 23, much smaller than the file size. We explore the initial advertised receive window size in all TCP flows, and observe that the values fall in 131,712$\pm$600 bytes for over 99% of the flows, for iOS, Android and Windows Phone devices. Second, the application is not reading the data fast enough from the receiving buffer at TCP layer. Otherwise, even if the initial receive window is small, the receive window size should not drop to close to 0 afterwards.

We further study the prevalence of such poor network performance throttled by the TCP receive window. We find that for all downlink TCP flows, 52.6% of them experience full receive window. And for 91.2% of these affected flows, the receive window bottleneck happens in the initial 10% of the flow duration. These observations suggest that about half of the TCP flows in the dataset are experiencing degraded network performance limited by the receive window size.

We also observe that some apps under-utilize the bandwidth due to the application design. Figure 24 shows the network behavior of the popular Netflix app [1] on iOS. The upper half of the figure shows the HTTP requests and responses on different client ports. At around 70s, the user browses through a list of video thumbnails and switches to another video. We observe that all HTTP requests for video download are HTTP byte-range requests and the corresponding responses are mostly short in duration (smaller than 1s, making them barely visible). The response sizes range from 1MB to 4MB. The client *periodically* requests for video chucks every 10s, with each TCP connection typically reused by two consecutive requests. The bottom half of the figure plots the aggregated downlink throughput, showing a clear periodic pattern corresponding to the periodic requests. While the peak throughput can reach up to 30Mbps, for most of the time, the network interface is idle. This type of traffic pattern is known for incurring high tail energy [24]. In this particular case, we know that the tail timer for the studied network is 10s, and based on the LTE radio resource control (RRC) state machine [13], the 10-second request periodicity would keep

the UE radio interface always at the high-power state, incurring unnecessarily high energy overheads.

7.3 Discussions

We have shown that multimedia traffic is dominant in LTE networks and the available bandwidth is far from being effectively utilized by many popular apps. Hence optimizing the network utilization for these apps is critical for improved user experiences and battery life.

For the TCP receive window problem, existing studies [16] have shown that smartphone vendors may have been reducing receive window sizes to mitigate the "buffer bloat" problem, resulting in TCP performance degradation. Dynamic receive window adjustment (DRWA) [16] is proposed to address this issue. However, such proposals require changes to TCP stacks, making their deployment potentially challenging. As an orthogonal solution, applications should read downloaded data from TCP's receiver buffer quickly. For example, the desired behavior of the Shazam player (§7.2) is to download the file as fast as possible, promptly move the data to application-layer buffers, and close the connection immediately when the file transfer is complete. Doing so benefits for both the network and device energy efficiency.

Regarding to the periodical network activities of the Netflix player (Figure 24), in addition to leveraging the application-layer buffer, it is also recommended that it send fewer requests and download more content for each request. Huang *et al.* [13] have shown that transferring data in a large batch significantly reduces the radio energy than otherwise. This also allows TCP to make better use of the available bandwidth.

8. CONCLUSION

In this paper, we use a large-scale LTE data set to study the impact of protocol and application behaviors on the network performance. We observe that some TCP behaviors, such as not updating RTT estimation using duplicate ACKs, can cause severe performance issues in LTE networks upon a single packet loss. By devising a novel bandwidth estimation algorithm, we observe that for 71.3% of the large flows, the bandwidth utilization ratio is below 50%. We also show that the available bandwidth for LTE networks has high variation and TCP is not able to fully utilize the bandwidth as the congestion window cannot adapt fast enough, especially when RTT is large. We further notice that the limited receive window size throttles the TCP performance for 52.6% of the downlink flows. In addition, we find that the application design may result in under-utilized bandwidth. All these findings provide insights on developing transport protocol mechanisms and applications that are more LTE-friendly.

9. ACKNOWLEDGEMENTS

We thank Professor Elizabeth Belding for her constructive comments serving as the shepherd for this paper. We also thank the anonymous reviewers for their feedback. This research was supported in part by the National Science Foundation under grants CNS-0643612, CNS-1039657, CNS-1059372 and CNS-0964545.

10. REFERENCES

[1] Netflix App. `http://www.netflix.com/`.

[2] Shazam App. `http://www.shazam.com/`.

[3] M. Allman, V. Paxson, and E. Blanton. Tcp congestion control. RFC 5681, 2009.

[4] M. Balakrishnan, I. Mohomed, and V. Ramasubramanian. Where's That Phone?: Geolocating IP Addresses on 3G Networks. In *Proceedings of IMC*, 2009.

[5] L. Brakmo and L. Peterson. TCP Vegas: end to end congestion avoidance on a global Internet. *Selected Areas in Communications, IEEE Journal on*, 13(8):1465 –1480, 1995.

[6] X. Chen, R. Jin, K. Suh, B. Wang, and W. Wei. Network Performance of Smart Mobile Handhelds in a University Campus WiFi Network. In *IMC*, 2012.

[7] J. Erman, A. Gerber, K. Ramakrishnan, S. Sen, and O. Spatscheck. Over The Top Video: The Gorilla in Cellular Networks. In *IMC*, 2011.

[8] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, and R. G. D. Estrin. Diversity in Smartphone Usage. In *MobiSys*, 2010.

[9] A. Gember, A. Anand, and A. Akella. A Comparative Study of Handheld and Non-Handheld Traffic in Campus Wi-Fi Networks. In *PAM*, 2011.

[10] A. Gerber, J. Pang, O. Spatscheck, and S. Venkataraman. Speed Testing without Speed Tests: Estimating Achievable Download Speed from Passive Measurements. In *IMC*, 2010.

[11] E. Halepovic, J. Pang, and O. Spatscheck. Can you GET Me Now? Estimating the Time-to-First-Byte of HTTP Transactions with Passive Measurements. In *IMC*, 2012.

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

[13] 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 *MobiSys*, 2012.

[14] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and P. Bahl. Anatomizing Application Performance Differences on Smartphones. In *MobiSys*, 2010.

[15] M. Jain and C. Dovrolis. End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. In *IEEE Network*, 2003.

[16] H. Jiang, Y. Wang, K. Lee, and I. Rhee. Tackling Bufferbloat in 3G/4G Networks. In *IMC*, 2012.

[17] X. Liu, A. Sridharan, S. Machiraju, M. Seshadri, and H. Zang. Experiences in a 3G Network: Interplay between the Wireless Channel and Applications. In *MOBICOM*, 2008.

[18] M. Mathis and J. Mahdavi and S. Floyd and A. Romanow. TCP Selective Acknowledgment Options. RFC 2018, 1996.

[19] G. Maier, F. Schneider, and A. Feldmann. A First Look at Mobile Hand-held Device Traffic. In *PAM*, 2010.

[20] V. Paxson, M. Allman, J. Chu, and M. Sargent. Computing tcp's retransmission timer. RFC 6298, 2011.

[21] R. Prasad, C. Dovrolis, M. Murray, and kc claffy. Bandwidth Estimation: Metrics, Measurement Techniques, and Tools. In *IEEE Network*, 2003.

[22] F. Qian, A. Gerber, Z. M. Mao, S. Sen, O. Spatscheck, and W. Willinger. TCP Revisited: A Fresh Look at TCP in the Wild. In *IMC*, 2009.

[23] F. Qian, J. Huang, J. Erman, Z. M. Mao, S. Sen, and O. Spatscheck. How to Reduce Smartphone Traffic Volume by 30%? In *PAM*, 2013.

[24] F. Qian, Z. Wang, Y. Gao, J. Huang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Periodic Transfers in Mobile Applications: Network-wide Origin, Impact, and Optimization. In *World Wide Web*, 2012.

[25] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Characterizing Radio Resource Allocation for 3G Networks. In *IMC*, 2010.

[26] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Profiling Resource Usage for Mobile Applications: a Cross-layer Approach. In *MobiSys*, 2011.

[27] R. Braden. Requirements for Internet Hosts – Communication Layers. RFC 1122, 1989.

[28] P. Sarolahti and A. Kuznetsov. Congestion Control in Linux TCP. In *USENIX Annual Technical Conference*, 2002.

[29] S. Sesia, I. Toufik, and M. Baker. LTE: The UMTS Long Term Evolution From Theory to Practice. John Wiley and Sons, Inc., 2009.

[30] C. Shepard, A. Rahmati, C. Tossell, L. Zhong, and P. Kortum. LiveLab: Measuring Wireless Networks and Smartphone Users in the Field. In *HotMetrics*, 2010.

[31] J. Sommers and P. Barford. Cell vs. WiFi: On the Performance of Metro Area Mobile Connections. In *IMC*, 2012.

[32] V. Jacobson and R. Braden and D. Borman. TCP Extensions for High Performance. RFC 1323, 1992.

[33] Z. Wang, Z. Qian, Q. Xu, Z. M. Mao, and M. Zhang. An Untold Story of Middleboxes in Cellular Networks. In *SIGCOMM*, 2011.

[34] Q. Xu, J. Erman, A. Gerber, Z. M. Mao, J. Pang, and S. Venkataraman. Identifying Diverse Usage Behaviors of Smartphone Apps. In *IMC*, 2011.

[35] Q. Xu, J. Huang, Z. Wang, F. Qian, A. Gerber, and Z. M. Mao. Cellular Data Network Infrastructure Characterization and Implication on Mobile Content Placement. In *SIGMETRICS*, 2011.

[36] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the Characteristics and Origins of Internet Flow Rates. In *SIGCOMM*, 2002.

[37] Z. Zhuang, T.-Y. Chang, R. Sivakumar, and A. Velayutham. A3: Application-Aware Acceleration for Wireless Data Networks. In *MOBICOM*, 2006.