Accelerating Mobile Web Loading Using Cellular Link Information

Xiufeng Xie
University of Wisconsin-Madison
xiufeng@ece.wisc.edu

Xinyu Zhang
University of Wisconsin-Madison
xyzhang@ece.wisc.edu

Shilin Zhu
University of Wisconsin-Madison
szhu@ece.wisc.edu

ABSTRACT

Despite the 4G LTE's 10× capacity improvement over 3G, mobile Web loading latency remains a major issue that hampers user experience. The root cause lies in the inefficient transport-layer that underutilizes LTE capacity, due to high channel dynamics, wireless link losses, and insufficient application traffic to propel the bandwidth probing. In this paper, we propose Cellular Link-Aware Web loading (CLAW), which boosts mobile Web loading using a physical-layer informed transport protocol. CLAW harnesses the limited PHY-layer statistics available on LTE phones to quantitatively model the LTE channel resource utilization, which is then translated into a transport window that best fits the bandwidth. Consequently, CLAW can estimate and fully utilize the available bandwidth almost within one RTT. In addition, CLAW can precisely differentiate LTE wireless loss from congestion loss, and identify the rare cases when the wireline backhaul becomes the bottleneck. We have prototyped CLAW on commodity LTE phones. Across a wide range of experimental settings, CLAW consistently reduces Web loading latency by more than 30%, compared to classical TCP variants and state-of-the-art congestion controls for cellular networks.

1. INTRODUCTION

Web access has become a main portal of content consumption on mobile devices, and user-perceived Web performance becomes a vital metric for network services. In online shopping, for example, merely 100ms increase in Web loading latency leads to 1% profit loss [1]. Over the past years, wireless service providers have strived to improve the cellular network performance to match the demanding Web access. However, the mobile Web loading performance does not match up to the infrastructure upgrade. Nation-wide 4G vs. 3G speed tests in the US showed 10× throughput improvement [2], but the typical Web latency only decreases by around half [3]. Given the typical mobile Web size of below 1.5 MB, and median 4G LTE throughput of 12 Mbps [2–4], one would expect the Web loading latency to be well below 1 second. In reality, most mobile webpages take 2 to 9 seconds to be loaded over LTE [3].

The disappointing mobile Web performance roots in the poor interaction between HTTP–the application-layer protocol for the Web, and TCP–the underlying transport-layer protocol. It is well established that TCP performs best when there is a steady stream of data packets and returning ACKs to keep the network pipeline full. This is because TCP implicitly uses the data packets to probe the network, and it takes time to gradually ramp up the data load towards the network's available bandwidth. Since HTTP generates short, bursty flows, TCP lacks time to converge, thus severely underutilizing the network capacity. Such laggy responses are exacerbated in cellular networks with relatively long RTT and dynamically changing bandwidth. In addition, existing TCP protocols use packet delay or loss as congestion indicators to facilitate bandwidth probing. However, the unstable RTT makes the delay metric unreliable and vulnerable to spurious timeout [3]. The loss metric, on the other hand, tends to overreact to wireless link losses, and is insensitive to real congestion because of the deep buffers at LTE basestations [5–7].

Given the mismatch between TCP's continuous packet flow requirement and HTTP's bursty traffic pattern, we propose to replace TCP's probing mechanism with a direct estimate of bandwidth from the LTE physical layer. This principle builds on several observations: *(i)* The end-to-end path for most cellular connections is bottlenecked by the last hop [8], particularly because content delivery networks (CDN) and Web caches shorten the wireline path. Hence it suffices to track the cellular link bandwidth, and gracefully detect and handle the corner cases when the wireline backbone becomes the bottleneck. *(ii)* Since an LTE basestation (BS) itself already enforces fair scheduling [9], the transport-layer protocol should focus on fully utilizing the available physical bandwidth. *(iii)* LTE BS's available frequency/time resources are shared among clients. Thus, it is feasible for a client to estimate the resource utilization within the shared medium from its PHY layer signaling with the basestation.

We realize these principles through a novel system design called Cellular Link-Aware Web loading (CLAW). CLAW accelerates mobile Web loading using a bandwidth exploration mechanism facilitated by cellular link statistics available on commodity smartphones. It replaces TCP's AIMD-style rate control with a PHY-layer informed scheme that can estimate available bandwidth almost within one RTT, without relying on the application traffic pattern. The key challenge here lies in the limited PHY information available to clients. Ideally a client would need a global view of the BS's resource usage, so as to estimate the amount of resources and hence bandwidth available to it. But the PHY signaling channel is encrypted and each user equipment (UE) can only observe the resource allocated to itself [10,11]. Although the BS could separate a dedicated control channel to broadcast the resource usage as public

MobiSys '17, June 19–23, 2017, Niagara Falls, NY, USA.
© 2017 ACM. ISBN 978-1-4503-4928-4/17/06...$15.00
DOI: http://dx.doi.org/10.1145/3081333.3081367

information, this requires non-trivial update to the LTE MAC/PHY stack and causes huge compatibility issue.

To overcome the constraints, we design a cellular load analyzer incorporating a quantitative model that allows an LTE phone to extrapolate cell-wide resource utilization, by leveraging public PHY statistics of the downlink channel. Prior work in LTE resource analysis needs fine-grained per-subcarrier channel information [10, 11], which can be obtained only through a software-radio monitor. Even if such information eventually becomes available, it entails non-trivial signaling overhead when streamed in real-time from the LTE modem and to user space. In contrast, our model enables accurate cell load analysis even with abstracted diagnostic information (*e.g.*, average signal power and modulation level), which is readily available on commodity LTE phones.

Given the estimation of available frequency/time on the PHY layer, CLAW further translates it into usable bandwidth, and feeds it back to facilitate rate control at the TCP sender (*i.e.*, Web server). Unlike prior LTE resource analyzers [10–12], CLAW can estimate both cell-wide and individual client's resource usage, enabling more informed rate control under competing traffic. Moreover, a CLAW client harnesses its link-layer retransmission and queue overflow statistics to discriminate wireless loss from real congestion, preventing unnecessary sender rate fallback.

In addition, CLAW incorporates a simple bottleneck detector, which identifies the rare cases of wireline bottleneck, through a consistency check between the PHY resource utilization and transport-layer window evolution. The detection does not require any dedicated probing traffic and can complete within a few RTTs. When a wireline bottleneck occurs, CLAW falls back to an AIMD-style bandwidth exploration, which resembles legacy TCP but still harnesses certain PHY information such as packet losses and usable PHY resources.

We have implemented CLAW on an Android client that executes the idle resource estimation and bottleneck identification, and an HTTP server that replaces legacy TCP's rate control with CLAW's cellular-informed rate control. We benchmark CLAW's performance against conventional TCP variants, along with two state-of-the-art transport protocols, CQIC [12] and Verus [13], which are designed for cellular networks. A thorough evaluation demonstrates that CLAW can quickly track the available bandwidth resource, leading to 27% to 66% latency reduction across a wide range of websites and network conditions. We also observe that loss/delay based TCP protocols often lead to large performance variation, and CLAW reduces the worst-case latency (which is likely to be the bottleneck to user experience) by up to 73%.

To our knowledge, CLAW represents the first work to quantitatively model the relation between LTE's resource utilization and the PHY-layer statistics on commodity cellphones. The model can be adapted by a broader range of wireless protocols, including not only web loading, but also video streaming [10], background traffic scheduling [14], *etc.* CLAW is also the first to achieve zero-overhead bottleneck detection and explicit loss differentiation over LTE. The CLAW design relies on PHY information, but entails no PHY-layer modification and is readily deployable on current LTE devices.

2. BACKGROUND AND RELATED WORK

Web loading is a sophisticated procedure involving both networking and computation. In general, it begins by downloading an HTML page which refers to the objects (*e.g.*, images) within the requested webpage, the browser then iteratively parses the HTML file while downloading the referred objects. In parallel with the downloading process, the client device progressively renders the retrieved objects on its display. Page loading time (PLT) is typically used as the performance metric, which accounts for both object downloading and processing time, and directly impacts end-user experience. Optimizations for both the downloading and computation are vital to enhancing the mobile Web performance. This paper focus on the networking part, we further discuss about this in §7.

HTTP has been the de facto vehicle protocol to deliver Web content, accounting for more than 82% of the traffic that ends at mobile devices [15]. Many of today's Web services adopt HTTP 1.1, which use short-lived TCP connections to deliver objects, one connection per object. This inevitably incurs long latency as modern Web sites contain tens to hundred level objects. The HTTP/2, inspired by Google's SPDY [3] and ratified by the IETF in 2015 [16], optimized the data transfer paradigms, allowing multiple outstanding object requests through one TCP connection. To date, HTTP/2 has been supported by all main-stream mobile browsers and content providers. However, recent measurement studies revealed disappointing HTTP/2 performance in cellular networks [3], due to *poor interaction between TCP, HTTP and the radio link layer*. In particular, TCP cannot track the high RTT variations and often treats an RTT surge as timeout, which in turn leads to spurious retransmissions and extended Web loading time.

Parallel to the protocol evolution, Web caching has been widely deployed through CDNs at the network edge. Browsers and intermediate DNS servers employ caching/prefetching to reduce DNS lookup latency, with up to 80% hit rate [17]. In addition, LTE cellular networks have widely adopted middleboxes that take over the TCP establishment and termination to reduce TCP's handshake latency [6]. All of these optimizations amortize the HTTP initialization latency and signifies the importance of reducing actual downloading time.

Our CLAW design builds on prior measurement insights which revealed that mobile Webs' performance bottleneck lies in the long RTT [18]. Since cellular networks' long RTT roots in the legacy infrastructure [19], the key to accelerating Web download is to reduce the number of RTTs needed to retrieve the objects.

The key to cutting the Web loading round-trips lies in the underlying transport layer, *e.g.*, HTTP/2 adopts the innovation to reuse a conventional TCP connection. However, conventional TCP protocols treat the network path as a black box, and attempt to infer network congestion from end-to-end metrics like delay and loss, which have to be generated through trial-and-error probing across many round-trips and hence lack responsiveness. In effect, most Web loading sessions are short and finish well before TCP converges to the network bandwidth [20]. Explicit congestion control protocols [21, 22] advocate more informed rate adaptation based on router's feedback. But the deployment is rare to date, as they require new router hardware and sophisticated queue management. CLAW can be considered as providing a better-informed explicit congestion feedback that quantitatively characterizes how much more traffic load can be added before it congests the network. Yet it is much easier to deploy, as it harnesses the cellular bottleneck, and only requires the client to act as a single-point rate estimator.

Recently, the new challenges in cellular networks like buffer bloat [7] and high link dynamics triggered a revisit of the transport protocol design. Sprout [5] uses a stochastic model estimated from throughput probing to forecast network bandwidth. Verus [13] adapts the sending rate following a delay-to-bandwidth mapping learned from a training phase. PCC [23] directly tests a wide range of sending rates and then picks the rate with the best measured performance. Although these model-driven protocols showed high throughput for bulk data transfer, they are unsuitable for mobile Web applications, because the short HTTP flows preclude building/updating/reusing an empirical model.

A vast body of research has touched upon cross-layer transport protocols for wireless networks [24, 25], but most built on abstract models. Recently, CQIC [12] built a transport protocol atop UDP, which adapts rate by estimating link capacity in HSPA+ cellular networks. However, a CQIC client cannot perform bandwidth exploration – it essentially estimates the bit-rate supported by the fraction of channel time it is *currently allocated*, but is blind to the idle time/frequency resources that it can further *explore*, which results in an over-conservative bandwidth estimation. LoadSense [14] takes PHY layer metrics at coarse time granularity (5 per second) as input, and empirically classify the cell load as a binary "busy" or "idle" using a support vector machine. It then sneaks traffic into the idle period to improve energy efficiency. CLAW, on the other hand, models the cell load quantitatively and at millisecond resolution. piStream [10] uses a software-radio board to monitor the LTE BS's resource usage which in turn guides video applications' rate adaptation. In contrast, CLAW works for commodity LTE phones, and develops a new model to estimate the resource usage based on the limited PHY information available on LTE phones' diagnostic interface. Eventually, the per-subcarrier channel information needed in the piStream model may become available just as in certain WiFi chipset, but streaming such information from the LTE modem to the user space at the granularity of LTE frames still entails non-trivial overhead. Moreover, piStream cannot discriminate a client's own resource usage from competing users, leaving fairness an open issue (Sec. 4.3.1).

Cellular last-mile typically bottlenecks the end-to-end connection. However, in some corner cases, the bottleneck may appear in the wireline path and the server can adapt correspondingly for performance optimization [26]. State-of-the-art bottleneck identification schemes like Qprobe [27] require massive dedicated probing packets, which makes them infeasible in short-flow applications as the probing traffic itself overshadows data transmissions. In contrast, CLAW employs a bottleneck detector facilitated by the awareness of cellular network load, which bears zero overhead, can run in parallel with Web data transfer and discriminate between wireline and cellular bottlenecks in real time.

3. CHALLENGES FOR WEB OVER LTE

In this section, we present the root causes to the unsatisfactory mobile Web performance in LTE networks, and highlight the challenges in resolving these issues.

The large and unstable RTTs mislead TCP adaptation. Although LTE adopts a more flattened network architecture and reduced the number of gateways compared with 3G, its 70ms average TCP handshake RTT remains much longer than wireline networks and even WiFi [2], and the RTT over LTE quickly inflates with the number of bytes in flight (it easily exceeds 100ms with only 100KB data in flight). In addition, LTE links often experience large RTT variation due to link-layer packet retransmission/reordering, channel quality change, and traffic dynamics of competing LTE clients. The RTT's standard deviation is typically $3\times$ to $4\times$ of the mean for static LTE nodes, and even larger for mobile ones [19]. Such large RTT variations lead to the poor performance of conventional TCP during mobile Web loading (Sec. 6), which falsely treats a temporary RTT surge as packet timeout caused by network congestion.

Fig. 1 shows one case study, where we load the same static webpage 50 times on an LTE phone (detailed page hosting setup is in Sec. 6). The loading events scatter sparsely across the day to include various network conditions. The server uses the default TCP Cubic, and each page loading involves around 2700 packet transfers. From the results, we observe that packet losses are rare over LTE networks. However, even a small number of lost packets can significantly slow down the Web loading, which is consistent with existing observations [1]. Furthermore, under few or no packet losses, sometimes the Web latency still surges due to TCP's aforementioned sensitivity to RTT variations: a temporary RTT surge misleads the server to assume a network congestion/timeout and exit TCP slow start early, even when its current sending rate is still far below the optimum. To minimize such negative effects caused by cellular RTTs, *an efficient transport protocol should be able to quickly converge to the network bandwidth, with least number of round-trip probes.* Quick convergence also enables accurate adaptation to LTE channel dynamics, preventing bandwidth underutilization (due to underestimation) and self-inflicted congestion (due to bandwidth overestimation).

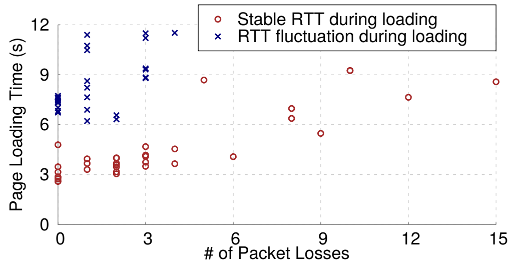


Figure 1: Page loading latency over LTE is affected by both RTT and packet losses.

TCP overreacts to LTE link losses. Although LTE's link-level retransmission improves reliability, packet losses are still unavoidable, and even a few losses across an entire Web loading session can substantially increase latency [28]. This is because TCP protocols typically treat packet loss as a congestion indicator, and multiplicatively reduce the sending rate even when the loss is caused by fluctuations of wireless link quality rather than a network congestion, resulting in poor bandwidth utilization and hence slower flows. As shown in Fig. 1, across the experiments, we observed very low packet loss rate, less than 1% even in the worst case. But we found even a few more packet losses lead to explosive growth of latency, which echoes the large-scale measurements in [28]. Moreover, due to unpredictable wireless losses, the Web latency becomes very unstable across runs. The worst-case latency can be extremely high and ruin the user experience, as the users are sensitive to performance degradations.

The short mobile Web flows hinders the sending rate from quick convergence to the network bandwidth. TCP's bandwidth exploration is propelled by application traffic. Its congestion window size accumulates as more packets are acknowledged over previous round-trip transmissions. This property fares poorly with HTTP that creates sparsely distributed packet bursts — measurements studies showed that 90% of Web flows finish within TCP's slow start phase [20], *i.e.*, well before it converges to network bandwidth.

To showcase the problem, we load a static HTTP/2 webpage on an LTE phone for 5 times consecutively with small time interval. Fig. 2 plots the loading time and evolution of TCP congestion window (`cwnd`) size. In the 1st trial, due to the slow start mechanism, TCP aggressively increases the `cwnd` by one upon receiving each ACK (acknowledgement) packet. The `cwnd` ideally doubles per RTT if it were for bulk transfer, but in fact accumulates slowly due to lack of sufficient HTTP data to fill in the window. In the following trials, since HTTP/2 reuses the TCP connection, the `cwnd` gradually accumulates and the 5th trial's latency almost reduces to $1/3$ of the 1st one. Obviously, TCP fails to fully explore the avail-

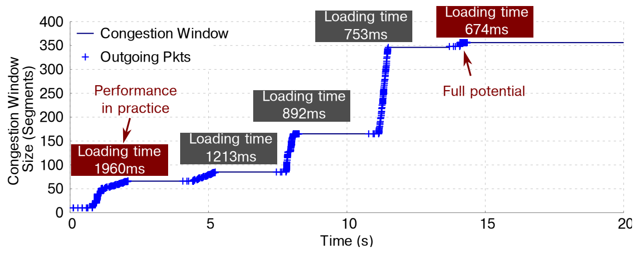


Figure 2: A snapshot of TCP sender cwnd for 5 back-to-back loading events visiting the same static HTTP/2 webpage (with browser cache disabled).

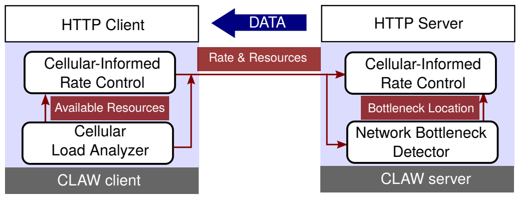


Figure 3: CLAW system architecture.

able bandwidth as mobile Web traffic lacks sufficient data to drive TCP's self-clocked rate adaptation. Thus, *an efficient rate control algorithm for mobile Web should be able to estimate the network bandwidth regardless of the source bit-rate.*

4. CLAW DESIGN

We propose Cellular-Link-Aware Web-loading (CLAW), which harnesses LTE's PHY-layer statistics, including signal energy, packet loss and modulation schemes, to accelerate the network bandwidth exploration for mobile Web traffic. CLAW aims to approach a design ideal that makes TCP converge to network bandwidth within one RTT. Below we outline CLAW's operations and components that work towards the design goal. CLAW consists of three mechanisms running on the HTTP client (*i.e.*, LTE phone) and server (Fig. 3). The CLAW client runs a *cellular load analyzer* that models the cell-wide downlink traffic load on BS and estimates available resources, based on the real-time PHY statistics locally available from the phone's diagnostic interface. The client then performs *cellular-informed rate control* to map available resources into additional traffic that can be carried over the network path, and accordingly optimizes the congestion window size to be fed back to the CLAW server. Meanwhile, the client employs a *cellular bottleneck identifier* to detect and adapt to the rare cases with wireline rather than LTE link being the network bottleneck. In what follows, we introduce CLAW's design components in detail.

4.1 A Primer on LTE Resource Structure

LTE BS allocates radio resources to clients in the unit of time and frequency blocks (Fig. 4). In time domain, the channel is divided into *frames* of 10ms length, each comprising 10 *subframes* of 1ms length. A subframe can be further divided into two 0.5ms *slots*. In frequency domain, LTE further uses the OFDM modulation to divide an entire spectrum band (5MHz or 10MHz) into small units. *Resource Element* (RE) is the smallest resource unit, which spans one OFDM symbol (66.7μs) in time and one OFDM subcarrier (15kHz) in frequency. *Resource Block* (RB) is the smallest *allocatable* resource unit. Typically, each RB contains 7 $\times$ 12 REs as it spans across 7 time-domain symbols (total 0.5ms, or one slot) and 12 frequency-domain subcarriers (total 180kHz).

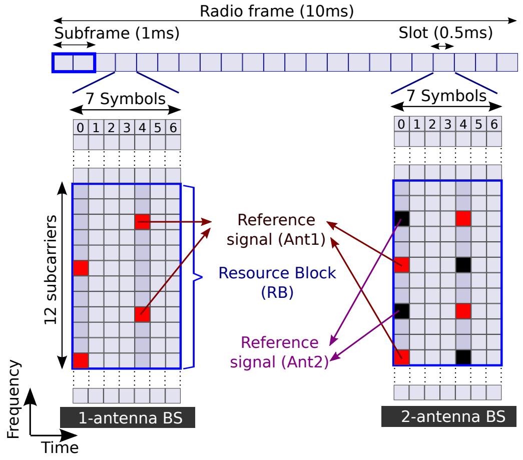


Figure 4: LTE downlink resource structure.

LTE clients can access the following cellular link statistics which characterize the quality of PHY resources:

Receive Strength Signal Indicator (RSSI): the total received power across the entire downlink frequency band, which is measured from the OFDM symbols carrying *reference signals* (symbol 0 & 4 as shown in Fig. 4).

Reference Signal Received Power (RSRP): average per-RE received power on the REs carrying reference signals, *i.e.*, the sum power of all reference REs divided by the number of reference REs.

Reference Signal Received Quality (RSRQ): the ratio between RSRP and RSSI, *i.e.*,

$$Q = N_{rb} P / I \tag{1}$$

where Q, P and I denote RSRQ, RSRP, and RSSI, respectively. N_{rb} is the number of RBs across the downlink spectrum band, specified to N_{rb} = 25 and 50, for 5MHz and 10MHz downlink band, respectively [29].

Modulation and Coding Scheme (MCS): an index to the combination of modulation type, coding rate and number of spatial streams (for MIMO transmission), which can be translated into bit-rate through a look-up table.

In addition, each client can read the BS's configuration parameters, such as frequency bandwidth and the number of antennas N_{ant}, from a shared control channel.

4.2 Cellular Load Analyzer

Instead of relying on the irresponsive network congestion indicators like packet delay/loss to probe the network capacity, CLAW can quantitatively assess the gap between current traffic rate and the network capacity based on the *cell load*, *i.e.*, the volume of allocated radio resources (time/frequency) on the downlink channel. It then translates the load estimation into a proper sending rate for the server that fully utilizes the network resource without causing congestion.

Ideally, the BS knows the cell load and can pass such information to all clients. But such a new signaling channel requires modifications to the LTE MAC/PHY standard. Alternatively, cell load information may be obtained if a client can decode the BS's downlink control channel which specifies the resource allocation. But the BS logically isolates the control channel among different clients [11], so that the standard LTE clients are blind to each other's resource usage. More specifically, on commodity smartphones, although the diagnostic interface can expose the per-subframe resource allocation statistics to the user space, it only shows the resource allo-

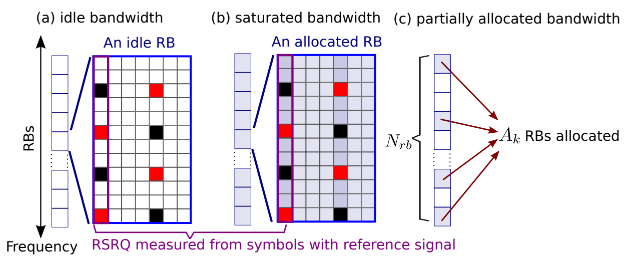


Figure 5: LTE reference signal under different resource allocation status (example shows a 2-antenna BS).

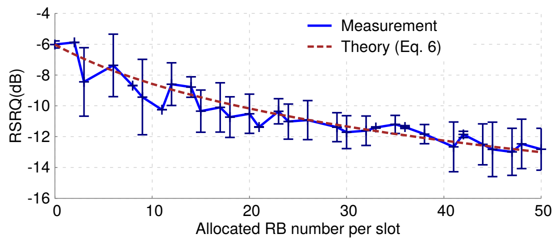


Figure 6: CLAW models LTE downlink resource allocation using a phone's locally sensed RSRQ metric. This figure compares the model with measurement (error bars show std.).

cated to the phone itself, while the cell-wide resource utilization information remains unavailable. Substantial hardware modifications or external sniffers [10,30] are needed to enable an LTE client to decode the downlink control channel dedicated for other clients. The aim of CLAW's cellular load monitor is to make precise cell load estimation by harnessing the PHY metrics available on unmodified commodity phones.

Quantitative cell load estimation based on RSRQ. The RSRQ metric is originally provided to LTE clients to assist local decisions like handoff and bit-rate adaptation. By digging deep into the LTE resource structure, CLAW repurposes RSRQ and quantitatively maps it to the cell load, *i.e.*, a BS' cell-wide downlink resource allocation. In what follows, we describe the analytical model for the mapping.

(i) When the downlink channel is idle, the client only receives reference signals, and thus the total received power over one RB is $N_{ref}P$, where N_{ref} is the number of REs occupied by reference signals in one RB, and P is the RSRP. Meanwhile, the RSSI is measured within the OFDM symbols carrying reference signals. Since the OFDM symbols span the entire bandwidth or N_{rb} RBs, the resulting RSSI is:

$$I_{idle} = N_{rb}N_{ref}P \tag{2}$$

Per LTE specification, $N_{ref} = 2$ and 4, for $N_{ant} = 1$ and 2, respectively, where N_{ant} denotes the number of antennas on the BS. Combining Eq. (1) and (2), the corresponding RSRQ equals $1/N_{ref}$, *i.e.*, 1/2 and 1/4 (or -3 dB and -6 dB), respectively.

(ii) Now we analyze a saturated downlink channel with full resource utilization. The reference signals from different BS antennas do not overlap [29], so the received power of an RE carrying reference signal still equals P, whereas the power on an RE carrying data becomes $N_{ant}P$. As one RB spans across $N^{rb}_{subc} = 12$ REs, the power of one RB equals $N_{ref}P + (N^{rb}_{subc} - N_{ref})N_{ant}P$, and the RSSI over the fully occupied bandwidth is:

$$I_{full} = N_{rb}\big(N_{ref} + (N^{rb}_{subc} - N_{ref})N_{ant}\big) \cdot P \tag{3}$$

From Eq. (1) and (3), we infer that on a saturated LTE channel, RSRQ= $1/12 \approx -11dB$ and RSRQ= $1/20 \approx -13dB$ for BS with $N_{ant} = 1$ and 2, respectively.

(iii) For a downlink channel with intermediate traffic load, consider a subframe k with A_k out of the total N_{rb} RBs allocated, the corresponding RSSI consists of the power over the A_k occupied RBs and $(N_{rb} - A_k)$ idle RBs:

$$I_k = A_k I_{full} + (N_{rb} - A_k)I_{idle} \tag{4}$$

Combining with Eq. (1), we have the RSRQ:

$$Q_k = \frac{N_{rb}P}{A_k I_{full} + (N_{rb} - A_k)I_{idle}} \tag{5}$$

Given I_{idle} from Eq. (2) and I_{full} from Eq. (3), we can derive the relation between Q_k, the RSRQ, and k, the number of allocated RBs in this subframe:

$$Q_k = \frac{1}{A_k(N^{rb}_{subc} - N_{ref})N_{ant}/N_{rb} + N_{ref}} \tag{6}$$

To validate this model, we record the number of allocated RBs for each subframe with an RSRQ measurement when performing web loading over LTE. Our result combines records from multiple BSs in our area, all of which have 10MHz bandwidth ($N_{rb} = 50$) and 2 antennas. Fig. 6 plots the average RSRQ value for each subframe-level RB allocation number. We can observe that the measurement results match the theoretical computation well. As a result, the fraction of allocated RBs in a time window, or cell load, can be reliably estimated based on the LTE-frame level RSRQ measurements following Eq. (6), which is the foundation of CLAW design.

Estimating available resources for a client. Following the model in Eq. (6), CLAW computes the instantaneous resource utilization, *i.e.*, fraction of allocated RBs in a subframe, as:

$$r_k = A_k/N_{rb} = \frac{Q_k^{-1} - N_{ref}}{(N^{rb}_{subc} - N_{ref})N_{ant}} \tag{7}$$

In practice, CLAW reads Q_k, N_{ant} from the phone's diagnostic interface, and all other parameters in Eq. (7) can be derived from the LTE standard, following our foregoing analysis.

To combat the burstiness in network traffic and the model variance shown in Fig. 6, CLAW accesses the current cell load L by averaging r_k across all K samples of RSRQ collected during a sliding *time window* of length T:

$$L = (\sum_{k=1}^{K} r_k)/K \tag{8}$$

An interesting fact is that if we set the window size T as the round-trip time (RTT), the traffic capacity of the currently unused radio resources in this window is exactly the gap between current TCP congestion window and its target value — the bandwidth-delay product. Therefore, CLAW uses RTT as its time window size T.

Given the cell load L, a CLAW client can gauge the *number of RBs allocated to all clients* within this time window T, as $N_a = N_t L$, where N_t denotes the total number of RBs within T. N_t can be easily obtained following the LTE specification: Since each subframe lasts for 1ms, the number of subframes in a time window equals its length T in milliseconds. Also, one subframe consists of two time slots, each containing N_{rb} RBs, so the total number of RBs within T equals $N_t = 2N_{rb}T$. Then the client can obtain N_I, the *number of idle RBs* within T:

$$N_I = N_t - N_a = 2N_{rb}T(1 - L) \tag{9}$$

Algorithm 1 Cellular-Informed Rate Control.

```
    Input: $\overline{\text{MCS}}$ – Average MCS in sliding time window
        $N_a$ – Sum number of RBs allocated to all clients
        $N_t$ – Total amount of RBs in the sliding time window
        $N_f$ – The number of RBs available to this CLAW client
        $\mathcal{P}$ – Bottleneck location (0 is cellular bottleneck)
    Output: cwnd – to be used by server
 1: /*CLAW Client*/
 2: Slide forward RTT-sized time window by one subframe
 3: Estimate BDP $\hat{B}$ based on $\overline{\text{MCS}}$ and $N_f$ following Eq. (12)
 4: The optimal congestion window $W_{claw} = \hat{B}$
 5: Feedback $W_{claw}$ to the CLAW server
 6: Repeat the steps above
 7: /*CLAW Server*/
 8: On receiving a new $W_{claw}$
 9: if ($\mathcal{P} == 0$) then /*Cellular bottleneck*/
10:     if $N_a/N_t < 1$ then
11:         /*Cellular-Informed BW Exploration*/
12:         cwnd $= W_{claw}$
13:     else/*On saturated channel, act as aggressive as Cubic*/
14:         cwnd $= max(W_{cubic}, W_{claw})$
15: else /*Wireline bottleneck*/
16:     cwnd $= W_{cubic}$
```

Denote the *number of RBs allocated to the client itself* as S_k, which can be read from the phone's diagnostic interface for every scheduled LTE subframe. At the end of a time window T, the client obtains N_s, the sum number of RBs allocated to itself within T:

$$N_s = \sum_{k=1}^{T} S_k \tag{10}$$

Finally, CLAW estimates N_f, the number of *available* RBs in this time window T, which consists of its currently consumed RBs and additional idle RBs it can leverage:

$$N_f = N_s + N_I \tag{11}$$

4.3 Cellular-Informed Rate Control

Given the estimation of currently available radio resources, CLAW performs cellular-informed rate control to shape the traffic bit-rate to the optimum by adapting the transport-layer congestion window (cwnd) size. Algorithm 1 outlines our design and we provide more details in what follows.

4.3.1 Cellular-Informed Bandwidth Exploration

Quick bandwidth convergence facilitated by cellular link statistics. During bandwidth exploration stage, CLAW boosts the server's sending rate to network capacity based on available radio resources. After estimating the volume of available RBs (N_f) in current RTT-sized time window T_i, the CLAW client translates N_f into the congestion window size matching the potential network capacity. Specifically, from the phone's diagnostic port, the CLAW client reads the Modulation and Coding Scheme (MCS) for every allocated LTE subframe. It then estimates the cellular link capacity $\hat{B}$ that can be supported by the N_f RBs under average MCS in current time window:

$$\hat{B} = T_i F(N_f/T_i, \overline{\text{MCS}}) \tag{12}$$

where the mapping rule $F(\cdot)$ is specified in the LTE standard [31, Table 7.1.7.1 and 7.1.7.2], which maps the MCS and the number of RBs per subframe (N_f/T_i) to the *Transport Block Size*, *i.e.*, the number of bytes that can be carried over a subframe. This size

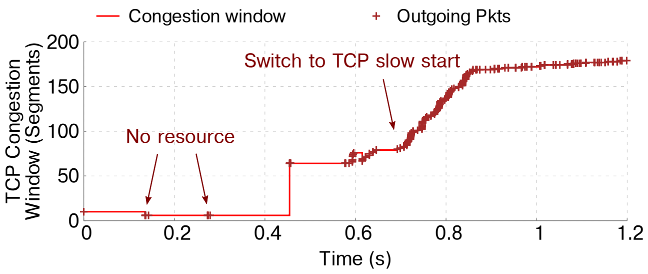


Figure 7: A running instance of the cellular-informed hybrid bandwidth exploration over an LTE channel saturated by existing users.

$F(N_f/T_i, \overline{\text{MCS}})$ is further multiplied by T_i, the number of subframes in a time window, to obtain the potential capacity $\hat{B}$. Since CLAW uses RTT as the time window length (§4.2), $\hat{B}$ essentially equals the bandwidth-delay-product, *i.e.*, the ideal congestion window size W_{claw} that best fits current network condition. W_{claw} is updated every time the sliding time window T moves forward, it is then fed back to the server, which sets W_{claw} as the congestion window (cwnd).

CLAW's cellular-informed bandwidth exploration design essentially addresses the conflict between the sparse Web traffic and the data-driven network capacity probing mechanism discussed in §3. Even with a small volume of traffic, CLAW can still instantaneously *converge to the available LTE bandwidth in one RTT*.

Hybrid bandwidth exploration mode. Note that a fully utilized LTE downlink channel does not necessarily imply the client has converged to its fair-share bandwidth. For instance, when the cellular downlink is saturated and all PHY resources are fully exploited, a new CLAW user that just joined the cell will keep estimating $W_{claw} = 0$, thus failing to explore the network bandwidth. To handle such situations, CLAW employs a *hybrid bandwidth exploration* mechanism, in which the sender maintains another window size W_{cubic} following the legacy TCP Cubic algorithm, and finally selects the larger one between W_{claw} and W_{cubic} as the actual cwnd. To avoid the false alarm of channel saturation caused by the traffic burstiness, we allow CLAW to switch to TCP Cubic only after it keeps detecting no idle resources for two consecutive RTTs, and then compensate it for the window size it could have increased in these two RTTs when it performs the mode switching. This hybrid bandwidth exploration design ensures that even over a saturated cellular link, the newly joined CLAW clients can still compete for the bandwidth as aggressively as the legacy TCP flows.

To demonstrate its effectiveness, we showcase a running instance of the hybrid bandwidth exploration in Fig. 7. We plot the server's cwnd and mark the sending time of each outgoing packet during a Web loading session over a saturated LTE channel. We see that the CLAW sending rate freezes in the beginning since there are no idle resource to explore. But after the 2-RTT inertia, mode switching is triggered: it first has a steep cwnd increase which follows the aforementioned window size compensation, and then ramps up with TCP Cubic's slow start operations.

Multi-user fairness. CLAW's cellular load awareness enables a client to detect the activities of competing downlink clients in the same cell, which allows it to explicitly achieve multi-user fairness. In contrast, conventional TCP is blind to competing clients and can slowly approach fair-share only for long-lived flows based on probing-and-trial. Specifically, CLAW design ensures fairness no matter when coexisting with conventional TCP flows or other CLAW flows: *(i)* When the LTE downlink channel is partially utilized by existing TCP Cubic or CLAW clients, a CLAW flow only takes its currently allocated resources plus idle resources. It does not attempt to preempt the resources currently occupied by other clients. *(ii)* Under a saturated LTE channel, CLAW's hybrid bandwidth exploration mode (Sec. 4.3.1) ensures it to be as fair as legacy TCP Cubic when competing for its fair share of bandwidth. *(iii)* When existing CLAW flows have fully exploited the downlink resources and other TCP Cubic/CLAW clients join later, the LTE BS will allocate less resources to existing flows based on its built-in fairness scheduling algorithm (proportional fairness for typical BS), thus an existing CLAW flow can promptly scale down its traffic rate following the reduced W_{claw} after detecting less fraction of resources allocated to itself, so as to prevent a potential network congestion. Then it falls back to TCP Cubic to guarantee the fairness afterwards. As a result, *CLAW achieves better fairness than conventional TCP since its resource awareness enables its rate adaptation to explicitly follow the BS's resource scheduling.*

4.3.2 *Discriminating Link Loss from Congestion*

Packet losses over cellular links, though infrequent, can severely disturb conventional TCP's congestion estimation, and lead to slow and unstable Web loading (§3). CLAW overcomes this fundamental limitation by discriminating the wireless loss and congestion loss using cellular link statistics available on commodity phones. LTE's RLC (Radio Link Control) layer handles the packet retransmission process on the downlink channel. The RLC layer statistics are available through the phone's diagnostic port, which include the number of packet losses, number of retransmitted packets and an overflow indicator for the RLC-layer packet queue. The CLAW client reports the RLC-layer loss events to the server, which then compare this information with the transport-layer packet losses. For instance, if a transport-layer loss happens along with RLC-layer losses caused by poor channel quality, the server notices it is not a congestion loss, and thus refrains from reducing the sending rate (congestion window). In contrast, the server will slow down immediately if the reported reason of packet loss is the buffer overflow at the LTE BS.

To facilitate the loss information feedback, CLAW marks the reserved bits in the header of the TCP ACK packets to represent different type of RLC loss events, so that the server can distinguish the wireless loss and congestion loss by reading these marks in the duplicate ACKs triggered by losses. §5 introduces how we implement this ACK flagging in the Android OS.

4.4 Network Bottleneck Detector

CLAW's cellular-informed rate control mechanism essentially boosts the traffic bit-rate to fit the client's share of cellular network capacity. However, in some corner cases, the network bottleneck may appear at the wireline path rather than the cellular link, and sending the traffic at the cellular capacity will lead to congestion at the bottleneck. To achieve a robust design, CLAW needs to detect and adapt to the rare scenarios where wireline bottlenecks the network.

Online bottleneck detection. CLAW's network bottleneck detector builds on a simple principle originating from its bandwidth exploration mechanism – If CLAW boosts the sending rate to harness currently unallocated resources, but the client's resource usage does not increase correspondingly, then there should be a bottleneck along the wireline path between the server and the BS.

To demonstrate the feasibility of this principle, we record the LTE resource allocation when loading the same Website, across a wireline bottleneck and cellular bottleneck, respectively (detailed

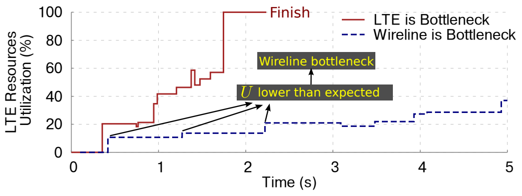


Figure 8: CLAW detects wireline bottleneck based on resource allocation pattern during bandwidth exploration.

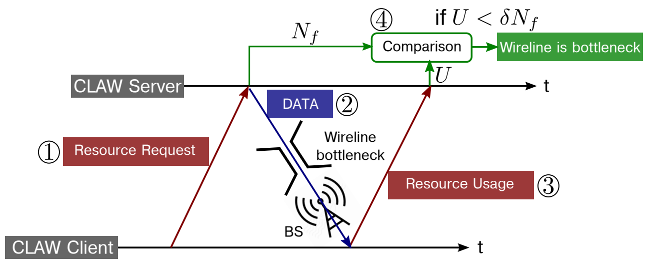


Figure 9: The network bottleneck detector in CLAW.

implementation in § 6.1). The results in Fig. 8 contrast the resource allocation patterns in the two cases, which show clear distinction.

Following this principle, the workflow of the network bottleneck detector design is outlined in Fig. 9. It runs in parallel with the cellular-informed rate control, at a time granularity of RTT level. Specifically, our design compares the volume of resources (N_f) available to a client, and its consequently allocated amount of resources (U) after one RTT. If $U < \delta N_f$, the detector declares a wireline bottleneck, otherwise a cellular bottleneck. The multiplicative factor 0< δ <1 controls the safe margin to accommodate for the RTT variations that can occasionally make U less than expected due to some delayed packets. We empirically set δ to 0.8. To further combat the network variations, the bottleneck is asserted only after consistent observations over a sliding window of 3 RTTs.

We emphasize that the bottleneck detector should be used as a sanity check. In most common cases, the wireless and wireline bottlenecks should have a non-trivial gap, even under channel variations, cross traffic disturbances, *etc.* So even a coarse check suffices. If these two happen alternately and inconsistently for a client, then they may be comparable and CLAW should degrade to the conventional TCP to avoid unnecessary oscillation.

Adapting to wireline bottleneck. After detecting a wireline bottleneck, CLAW falls back to TCP-style rate control. However, we emphasize that this fallback mode is still superior over conventional TCP, as it can still use cellular link statistics to discriminate wireless losses from congestion losses. Since the capability to discriminate between wireline and cellular bottlenecks is essentially a byproduct of the cellular-informed rate control and incurs zero overhead, CLAW client can run it in the background in parallel with web loading. Since it takes 3 RTTs to detect a wireline bottleneck, to avoid this latency, the client can also cache the decision based on server's IP address and reuse it for later sessions.

5. CLAW IMPLEMENTATION

Client side. Our implementation reads the logs containing the cellular link statistics from the diagnostic interface available on most cellphone models with Qualcomm chipsets. In Android system, the logs are passed to user space through /dev/diag, following the implementation of MobileInsight [32]. From such logs, we decode the RSRQ, MCS, PHY resource (RB) allocation, subframe index, and other statistics in real time with LTE subframe

(1ms) granularity. The output logs of the Qualcomm chipsets rearrange the cellular link statistics into obscure formats that can be decoded only by Qualcomm's QCAT, a proprietary software tool that parses the logs offline. To automate the real-time decoding, we reverse-engineer the log format by matching the raw HEX payload and QCAT's decoding results. We timestamp each decoded cellular link statistics based on which subframe it comes from, *e.g.*, the 7th subframe within the frame with System Frame Number (SFN) 100 has timestamp 107 (the elapsed milliseconds since the SFN counting started). Since the SFN cycles between 0 to 1023, we unwrap this value to avoid ambiguity. The decoded cellular link statistics are then fed to CLAW's cellular load analyzer implemented in the client's user space to facilitate the estimation of available resources following our model in § 4.2, and the estimation of the optimal congestion window size best fitting current network capacity following the algorithm in § 4.3.

In addition, the HTTP traffic pattern occasionally causes long idle durations, *e.g.*, when waiting for the response of HTTP GET, in which no downlink LTE traffic arrives. In such circumstances, the phone's radio enters the idle state and measures the downlink channel infrequently to save energy (only 5 RSRQ measurements per second), and hence is largely blind to the background traffic. Fortunately, since there is no traffic at all, the CLAW flow does not need to update the TCP congestion window. Therefore, the CLAW algorithm is frozen in such idle durations and takes effect only when a TCP connection is activated. In contrast, LoadSense [14] attempts to estimate the background traffic load before the client cuts into the cellular channel, where the radio is still in idle state and listens to the channel infrequently for every 200ms. Hence LoadSense can only get a rough correlation between the RSRQ and cell load, and a binary cell load estimation of "idle" or "busy". As a result, the on-the-fly operation of CLAW guarantees its more accurate cell load estimation over existing works.

For prototyping purpose, our current CLAW client implementation uses a UDP path to carry the optimal congestion window size for every 100 ms from the client to the server. Our future implementation will substitute this out-of-band feedback with an in-band signaling using the TCP Option field which can be more responsive. In addition, the client should also feedback the cellular-related packet loss events caused by BS buffer overflow or poor wireless channel (Sec. 4.3). To precisely synchronize the RLC-layer loss events reported by the diagnostic interface and the corresponding TCP packet loss, CLAW piggybacks the feedback in the TCP header like ECN-based TCP variants. To keep compatibility with current packet format, we use two of the 6 reserved bits in the TCP ACK's packet header (`th->res1`), one as the flag for the wireless channel loss, the other for the BS buffer overflow. To implement such a feedback channel in the TCP packet header, we modified `tcp_input.c` of the Android kernel to mark the wireless loss and buffer overflow notification bits, and `tcp_output.c` of the server kernel to extract these bits. We also add a customized `sysctl` variable to the Android kernel so that the CLAW application running in the user space can pass the identified loss & overflow events to the kernel.

Server side. We setup our test websites using an Apache HTTP/2 server [33] on an Amazon EC2 instance which also acts as the CLAW sender. At the server side, we tune the Linux TCP buffer size to guarantee the performance over high-speed networks like LTE. Besides, we set `net.ipv4.tcp_no_metrics_save` to 1 to ensure each Web loading test will not be affected by previous tests. To guarantee the performance of HTTP/2 which reuses TCP connection, `net.ipv4.tcp_slow_start_after_idle` is set to 0 as suggested in the HTTP/2 Wiki [34]. Upon receiving feedback containing the client's estimated `cwnd` size, the server passes it to CLAW's cellular informed rate control algorithm (Sec. 4.3), which runs in the user space and writes its `cwnd` size decision to the kernel via a customized `sysctl` interface. The new window takes effect by superseding the legacy TCP window just before the `cwnd` value is used for rate throttling within the packet transmission function `tcp_write_xmit()`. Such simple retrofitting allows flexible fallback to legacy TCP when CLAW identifies a wireline bottleneck (Sec. 4.4). For evaluation purpose, our modified server kernel also logs the packet losses, retransmissions, ACKs and behaviors of CLAW in real time. Since running CLAW implementation may change the TCP state parameters and HTTP/2 reuses TCP connection, to minimize the impact of the previous test to current Web loading, we reinitialize the TCP congest control state parameters by calling `tcp_init_congestion_control(sk)` in the TCP output engine at the end of each CLAW test. Overall, the majority of CLAW implementation lies in the client's user-space, and it only involves 41 lines of kernel patch at the server side.

6. EVALUATION

Experiment setup. We benchmark CLAW's performance against both classic TCP variants and state-of-the-art congestion control schemes for cellular networks, including CQIC [12] and Verus [13] which showed superior performance over alternatives (*e.g.*, Sprout [5]). Following [12], the CQIC implementation takes the LTE TBS (Sec. 4.3) controlled by the BS's MCS adaptation and resource allocation as the PHY capacity estimation. This essentially upgrades the original CQIC from HSPA+ to LTE [12]. We implement Verus in the server's Linux kernel following the source code (UDP-based) [35]. The original Verus updates `cwnd` every 5s, longer than a typical Web session. We thus reduce the time granularity to 500ms. Verus's bandwidth-delay profile is built during the slow start phase as suggested in [13]. For fair comparison, the tests with different schemes are conducted back-to-back to ensure similar channel conditions, and we employ the kernel's `sysctl` interface to enable fast and flexible switch between the algorithms.

Webpages and PLT measurement. We download and host a list of the Alexa top sites on an Amazon EC2 server that runs the HTTP/2 and CLAW's sender implementation. Following [4], the server hosts all Web objects of a website, including the dynamic objects generated by JavaScript, to avoid the clash between domain sharding and HTTP/2. More details about the sites we used can be found in §6.2.1. Our evaluation uses page loading time (PLT) as the primary performance metric, which directly impacts end-user experience. We use Chrome Developer Tools [36] which are widely adopted in Web loading related studies [37–39] to measure the PLT of a webpage over Android phones. The phones run the Chrome browser, with cache disabled to isolate individual Web loading tests.

Configurations for the phone. The smartphones used in our implementation are rooted so that the application layer can access and read from the diagnosis interface. We emphasize that no hardware modification is required, which is a big advantage over software-radio based cross-layer solutions like piStream [10]. Sometimes the phone may switch between LTE and HSPA when both networks are available, especially in the mobility experiments. Since the CLAW design is customized for LTE, we select "LTE Only" as the preferred network type inside the diagnostic and general setting mode of Android system to prevent unexpected network switching.

6.1 Micro-benchmarks of the CLAW design

Validating the cellular load analyzer. We first evaluate the ac-

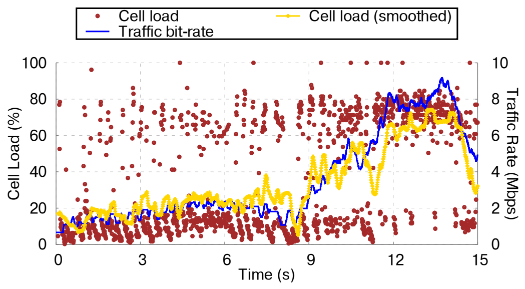


Figure 10: A microscopic demonstration of how CLAW's cell load analyzer can reliably track the cell load.

curacy of the cellular load analyzer and its real-time cell load tracking capability. We generate an `iperf` TCP traffic stream with bitrate increasing linearly by 0.5 Mbps per second, between the server and LTE phone. The experiment runs in late night isolate the interference from background traffic. Figure 10 plots the raw cell load (frame-level granularity) as scattered points, the smoothed cell load curve using a $200ms$ (approximately RTT sized) sliding window following Eq. (8), and the network layer traffic pattern measured at the phone using `tcpdump`. The results show that the raw cell load vary drastically over time, with a polarized resource allocation pattern – the BS either allocates most RBs in a subframe or leave it empty, implying that it enforces frame aggregation. After smoothing, we can observe a high consistency between the PHY-layer cell load estimated following §4.2 and the corresponding network layer traffic intensity, which verifies the accuracy of the cellular load analyzer design. In addition, this result demonstrates the feasibility of tracking cell load in real time using the phone's diagnostic interface.

Validating the cellular-informed rate control. To demonstrate the effectiveness of *Cellular-Informed Rate Control* in CLAW design, we load the same webpage using CLAW and TCP Cubic, the most widely deployed TCP variant. For fair comparison, we run the two experiments consecutively over the same idle LTE channel.

Fig. 11a plots the sender's `cwnd` over time. We see that CLAW explores the network bandwidth much faster than TCP Cubic. Once the data transfer starts, it first quickly maps the PHY resource to a close-to-optimum `cwnd`[1], and then keeps adapting the `cwnd` based on channel quality. Its downloading finishes early at 1.7s due to effective utilization of the bandwidth. In contrast, TCP Cubic can only gradually explore the bandwidth first via slow start, and then overshoots the bandwidth and jams the LTE downlink, because its congestion indicator, packet loss, is always masked by the large LTE buffer [40]. With queue building up at the BS, the RTT surges up and eventually exceeds RTO (Retransmission Timeout), which in turn causes multiplicative rate decrease. Such trial-based probing operations lead to long PLT (2.6s). Fig. 11a also plots one additional experiment that shows a case when Cubic's slow start fails due to RTT variation (§ 3), which causes early termination of slow start and hence even longer PLT (3.2s).

Fig. 11b further compares the corresponding resource utilization reported from the client. We can observe that the BS's PHY resource allocation follows the sender's traffic intensity, and CLAW exploits LTE resources much more efficiently than TCP Cubic.

To inspect CLAW's bandwidth utilization in comparison with other Linux TCP variants, we create a bare-bones HTML page with

[1]The sender side `cwnd` surge happens at around t=500ms, which exceeds one RTT. This issue is caused by the overhead to read the diagnostic interface, and process & feedback the computed `cwnd` in the application layer. A more refined implementation can further reduce such overhead and improve the responsiveness.

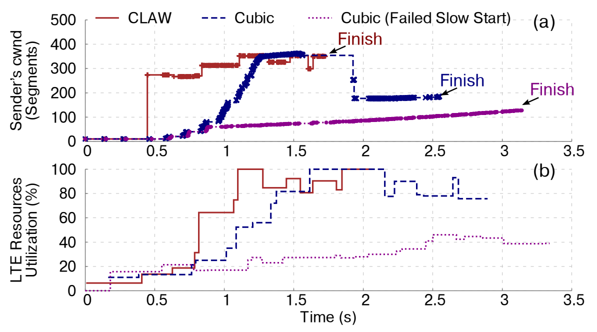


Figure 11: (a) CLAW server explores the network bandwidth faster than TCP Cubic (b) CLAW client is thus allocated more LTE PHY resources than TCP Cubic.

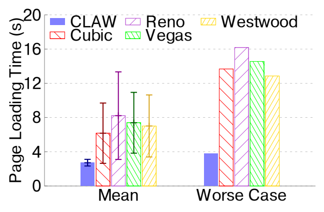


Figure 12: Web loading latency for a bare-bone Web page with a 2MB image.

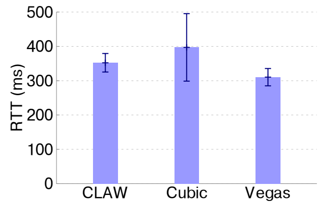


Figure 13: Packet latency (RTT) during loading Web page with a 2MB image

a single 2MB image, and load the page 20 consecutive times using each algorithm. Fig. 12 plots the average PLT and its standard deviation across 20 trials. CLAW obviously outperforms all the TCP variants and, compared to the runner-up algorithm TCP Cubic, it reduces the average PLT by 3.44s (56%). Remarkably, it also significantly reduces the PLT variance, reducing the worst case PLT by 9.92s (73%) compared with Cubic.

Fig. 13 further shows a microscopic inspection for the *packet-level* RTT of CLAW, along with the loss-based (Cubic) delay-based (Vegas) TCP. Since Vegas throttles rate on high RTT, it shows 48ms (15%) smaller mean RTT than CLAW, whereas interestingly its PLT is 2.2× longer! This is because Vegas often overreacts to delay spikes, resulting in extremely small queues and leaving the LTE BS barely anything to send. On the other hand, the BS's deep buffer makes Cubic insensitive to queue build up, misleading it to send at a high rate which causes self-inflicted congestion [5] and hence large RTT.

Effectiveness of CLAW's bottleneck detector. Since wireline bottleneck rarely occurs, to evaluate the bottleneck detector, we manually throttle the Web server's network interface at 30 Kbps using the `Dummynet` kernel module [41]. We run the same bare-bones page downloading experiment as above. Fig. 14 shows that, without the bottleneck detector, CLAW's performance becomes comparable to Cubic, because attempts to leverage the idle resources on

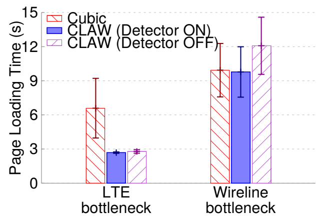


Figure 14: CLAW's bottleneck detector handles the rare wireline bottleneck.

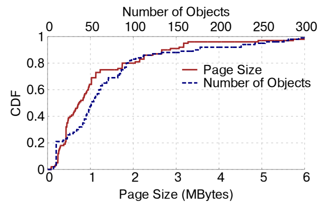


Figure 15: Profiling Alexa top 200 Website sizes.

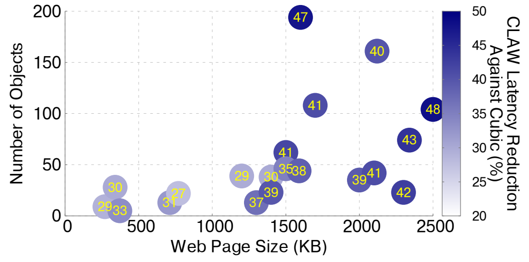


Figure 16: CLAW over different websites (number in circle is the percentage of latency reduction over Cubic).

the LTE channel when its sending rate already exceeds the wireline bottleneck capacity, which causes high RTT, packet loss or retransmission timeout. In contrast, we can see 2.3s (20%) latency reduction after turning on CLAW's bottleneck detector and fallback mechanism, which is also slightly better than the TCP Cubic performance since CLAW's fallback mode can still distinguish wireless losses. This experiment verifies that the bottleneck detector can effectively guarantee CLAW's performance in the rare case of wireline bottleneck.

6.2 System Level Test

In this section, we perform a system level test that compares CLAW's PLT against other transport protocols, across various Websites and network conditions. To control the experiment environment, unless otherwise stated, all tests load identical webpages on the same remote server. Each test loads the page for 20 times consecutively. The interval between adjacent tests is around 10s, long enough to ensure isolation and prevent HTTP/2 from reusing the previous `cwnd`.

6.2.1 *CLAW with Different Websites*

We conduct experiments with popular webpages selected from the top 200 Alexa list [42]. Fig. 15 plots the CDFs for both the page size and number of objects.

What kind of Webpages benefit the most from CLAW? We first evaluate CLAW's performance on 20 selected webpages that cover a wide spectrum of size and object number. To rule out the influence of other factors, we experiment on the idle LTE channel during late night at a static location, with other settings following the default configuration discussed at the beginning of this section. The scatter plot in Fig. 16 shows CLAW's PLT reduction *w.r.t.* TCP Cubic (in percentage) of each tested Website. Note that it is unlikely to have a page with small size but large number of objects, thus the upper-left corner of Fig. 16 remains blank.

We can observe that *(i)* CLAW reduces PLT by 27% to 48% compared with Cubic. *(ii)* CLAW's performance gain increases with larger page size or larger number of objects. Since CLAW accelerates page loading by faster downloading, it can save more time if the content downloading time accounts for more across the entire page loading process (Sec. 2). In addition, with more data packets, conventional TCP is more likely to experience at least one outage event, *e.g.*, overreacting to loss/RTT (§ 3), which causes extended PLT.

PLT under different transport protocols. We further compare CLAW and other benchmark protocols over 5 randomly selected popular webpages. From the results in Fig. 17, we can observe that *(i)* CLAW achieves the minimum latency and CQIC is always the runner-up. CQIC has around 50% higher PLT than CLAW because CQIC restricts itself to the currently allocated PHY resource, whereas CLAW effectively estimates the BS's remaining resources that it can exploit. *(ii)* Among the two delay-based congestion controls, Verus's PLT is even worse than TCP Vegas, because it is designed for long-lived flows, and cannot reliably learn the bandwidth-delay profile within the short HTTP flow.

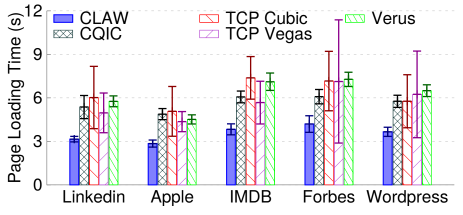


Figure 17: PLT under different transport protocols.

6.2.2 *CLAW under various network conditions*

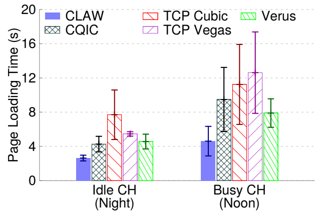
In the following set of experiments, we evaluate the impact of various network conditions to the page loading performance of CLAW and benchmark protocols. To isolate the influence of webpage size and object number, we test the same typical LinkdIn profile webpage (that people may need to frequently check over mobile phones) across the following experiment sets.

Under different LTE cell loads. To evaluate the impact of cell loads, we run the Web loading test at different time of the day – noon for a congested LTE cell, and late night for an idle cell. We used CLAW's cellular load analyzer to inspect the cell load before the tests and found that the background cell load is around 70% during the noon and close to 0% during late night.

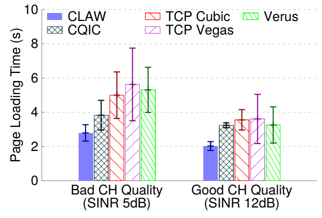
From the results in Fig. 18a, we make the following observations: *(i)* CLAW achieves the lowest PLT under different cell loads. For the idle channel, it reduces the latency by 5.06s (66%) against TCP Cubic and 1.63s (38%) against the performance runner-up, CQIC, while on the busy channel the reduction becomes 6.63s (59%) and 4.87s (51%), respectively. *(ii)* A CQIC client can only estimate the cellular link capacity supported by the fraction of resources allocated to itself [12], because it is unable to sense and harness the unallocated radio resources. This unfairly penalizes itself under a busy channel, leading to a sharp latency growth of 5.21s (55%) compared with the idle case. *(iii)* The delay-based TCP Vegas performs poorly over busy channel (mean PLT 8.04s), due to its well-known conservative rate control when coexisting with loss-based protocols [43]. Verus suffers from a similar problem, worsen by its inability to acquire a reliable bandwidth-delay profile.

Under different LTE channel qualities. We evaluate how the LTE channel quality affects the PLT by running tests at places with high (12dB SINR) and low (5dB SINR) channel quality reported by the phone. The results in Fig. 18b show that the PLT increases under lower channel quality for all schemes since the throughput unavoidably degrades and translates into longer downloading time. Yet, CLAW always achieves the lowest latency: it reduces the latency by 2.2s (44%) against the widely-deployed TCP Cubic and 1.04s (27%) against the performance runner-up, CQIC. Similar to CLAW, CQIC can also adjust the downstream rate following the channel quality, but it is incapable of exploiting unallocated idle resources (Sec. 2) and hence its performance always falls behind CLAW. Besides, lower channel quality tends to increase packet loss, but CLAW's PLT only increases by 0.76s, smaller than all other schemes, due to its capability of explicit loss differentiation (§ 4.3.2).

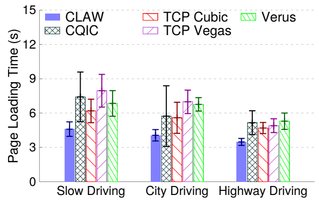
Under different outdoor mobility scenarios. We further run the tests in a moving vehicle under three outdoor mobility scenarios: *(i) slow driving* with around 10 mph within a university campus, *(ii) city driving* with around 20 mph in an urban area and *(iii) fast driving* with around 50 mph in a suburban highway. We care-



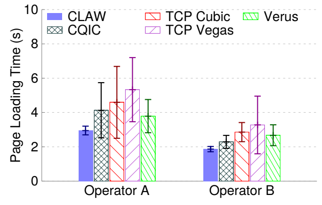
(a) Different LTE cell loads.



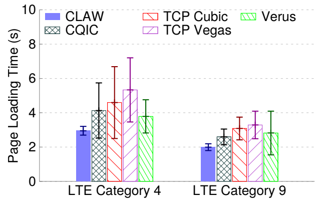
(b) Different LTE CH qualities.



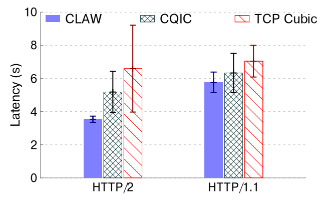
(c) Different mobility levels.



(d) Different LTE operators.



(e) Different phone models.



(f) Different web server.

Figure 18: CLAW outperforms benchmark schemes under various scenarios.

fully choose the spots on the route to perform Web loading so that the cross-cell handoff does not happen during a web loading.

From the results (Fig. 18c), we can observe that CLAW again achieves the lowest PLT across all the mobility scenarios. This proves that CLAW's cellular congestion monitor and rate controller perform consistently well even under very high channel dynamics. It may be counter-intuitive that the PLT decreases with the mobility level. We found the reason lies in the channel quality: the slow driving mode crosses routes between tall buildings which severely blocks the LTE signal; city driving partially suffers from similar effect; the highway driving occurs in relatively empty space with plenty of line-of-sights between the client and BS, resulting in extremely high link quality – our congestion monitor shows the corresponding TBS size is almost $3\times$ higher than the slow driving case.

Over different cellular operators. As a cross-validation, we run CLAW on the same phone supporting two operators possibly with different resource allocation policies. Fig. 18d shows that, despite the different average PLT due to different channel conditions, CLAW shows similar fraction of PLT reduction across the two operators, compared with alternative schemes. In particular, compared to TCP Cubic, CLAW reduces the page loading time by 1.6s (36%) for Operator A and 1.0s (35%) for Operator B. We also observe that Operator B provides better performance for all algorithms under the same channel quality, which owes to Operator B's BS scheduling algorithm that schedules the LTE client with shorter interval to propel the TCP self-clock. Regardless of the BSs used by different operators, CLAW achieves consistent performance gain over TCP Cubic since the BS's scheduling strategy equally impacts them. Due to lack of phone hardware, we are unable to test a wider range of operators. But we believe the experiment hints to the fact that the CLAW's design is not limited to a certain cellular operator.

Over different LTE hardware. To demonstrate that CLAW can always accelerate Web loading regardless of the peak LTE data rate

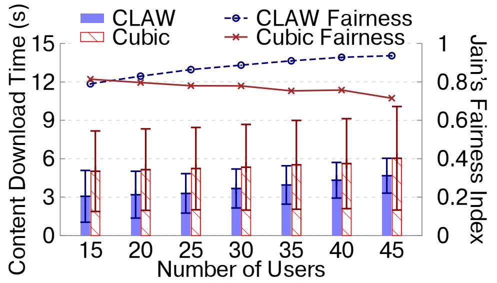
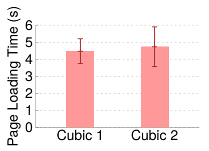
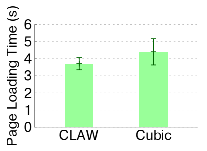


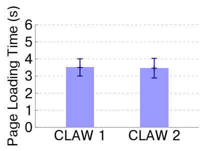
Figure 19: Scalability test.



(a) Two Cubic user load concurrently.



(b) CLAW&Cubic load concurrently.



(c) Two CLAW user load concurrently.

Figure 20: TCP coexistence and multi-user scenarios.

and processing power supported by the hardware, we cross-validate CLAW over two mainstream phone models: Nexus 5 with category [44] 4 LTE and Nexus 5X with category 9 LTE chipset, supporting 150 Mbps [45] and 450 Mbps peak downlink rate [46], respectively. The newer Nexus 5X is also equipped with more powerful hardware, *e.g.*, CPU, that accelerates the network stack and OS services. From Fig. 18e, we observe that the latter can significant accelerate page loading, due to higher PHY bit-rate and more powerful hardware. CLAW again shows consistent performance gain over both phone models. Interestingly, CLAW's latency reduction over TCP Cubic is around 35% for both Nexus 5 and Nexus 5X. In fact, although the low-end phones like Nexus 5 may lack proecessing power hence suffer from higher computational time, they typically have LTE hardware with lower category number or data rate. Therefore, reducing the network latency is still crucial for mobile Web acceleration.

With HTTP/1.1 and HTTP/2. Although CLAW is designed with HTTP/2 in mind, we have conducted experiments to verify its backward compatibility with HTTP/1.1. The same server is used to test different HTTP versions to isolate the influence of server hardware and the network path to the server. We switch the server between different HTTP versions by enabling/disabling the `http2` module of the Apache server. The results in Fig. 18f show worse performance for all tested algorithms when degrading the server from HTTP/2 to HTTP/1.1. In addition, compared to TCP Cubic, CLAW reduces the page loading time by 3.05s (46%) over HTTP/2 and a smaller 1.28s 18% over HTTP/1.1. This is because even though the network bandwidth explored by CLAW allows the Web contents to be downloaded all at once, without the multiplexing feature of HTTP/2, CLAW can only download the Web contents sequentially under HTTP/1.1, which raises the latency and reduces CLAW's performance gain.

6.2.3 *TCP friendliness and multi-user cases*

We conduct a microscopic examination of CLAW's TCP friendliness, by concurrently tapping the reload buttons of the same webpage on two phones of the same model, so as to create the worse-case traffic overlapping. We evaluate 3 configures with different combinations as shown in Fig. 20. By comparing Fig. 20a and Fig. 20b, we see that CLAW is as fair as TCP Cubic itself: when coexisting with CLAW, TCP Cubic achieves similar level of latency compared to when coexisting with another TCP Cubic flow.

Note that in the CLAW-Cubic coexisting scenarios, CLAW will not sacrifice its own performance for Cubic, rendering similar performance to the CLAW-CLAW coexisting scenarios in Fig. 20c, which owes to CLAW's gentle aggression – it exploits the idle resources without touching the resource occupied by other users, until the idle resources are drained off (§4.3).

Due to lack of LTE phones, we were unable to test a large number of CLAW users concurrently. Instead, we develop a simulator to simulate a network with LTE bottleneck. The BS uses a simple round-robin scheduler, and the PHY parameters, RTT, loss rate, are configured to be consistent with our empirical measurements. We implement and run CLAW and Cubic separately. Each of our experiments lasts for 30s with up to 45 users, wherein each user starts a 750KB page loading at a random time. The result (Fig. 19) shows that, the CLAW users consistently outperform Cubic even under a heavily loaded network. As the network saturates, CLAW's benefit diminishes as it has less idle resource to explore and falls back to the hybrid bandwidth exploration most of the time. Consistent with our reasoning in Sec. 4.3.1, CLAW shows a higher level of fairness (in Jain's index) than Cubic.

7. DISCUSSION

Comparison with BS-based solutions. CLAW works with commodity phones and existing LTE infrastructure, which makes it readily deployable. In contrast, redesigning the BS to explicitly convey bandwidth information, as suggested in [47], requires fundamental modifications to the LTE infrastructure, phone hardware and MAC/PHY standard. Furthermore, such solutions require close cooperation between content providers and the cellular operators, whereas a software-based solution like CLAW is more feasible for the vast majority of Web services like online retailers.

CLAW's design is also transferable to future-generation cellular networks, as long as the clients can collect certain PHY statistics as CLAW does for 4G LTE networks. Furthermore, future cellular networks may deploy more TCP proxies and caching servers to optimize the Web latency. CLAW could be the perfect match for these servers as the cellular link is almost surely the bottleneck in such scenarios.

Computational latency vs. networking latency. Recent research [48, 49] reveals that the computational time also plays an important role in the Web loading latency. However, even under the extreme cases with a tiny RTT of 5ms, the content downloading time still accounts for more than 40% of the total latency [49], and starts to dominate under higher RTT of 150ms. We emphasize that the RTT quickly inflates during data transfer due to network queuing [2, 5, 7, 13], and can easily exceed 150ms with only 200KB data in flight [2], which further justifies CLAW's strategy of accelerating the bandwidth convergence with fewer round-trips.

Generations of smartphones have been embracing escalating CPU power to minimize the computational latency, which always outpaces the network upgrades. In addition, to catch up with the rapid growth of phone screen size/resolution, the Web content sizes will scale accordingly to offer better user experience, which further elevates the networking latency. We believe that CLAW's data transport protocol can better prepare the mobile networks for future demands. It can certainly be combined with page optimization [50] and better object fetching strategies [37] to further cut the PLT.

Remarkably, with shorter PLT, a CLAW user's radio interface stays in the energy-hungry active state for shorter duration, which directly translates into energy saving. We also plan to explore this benefit in our future work.

8. CONCLUSION

In this paper, we have presented the design, implementation and evaluation of CLAW, which accelerates mobile Web loading through a physical-layer informed transport protocol. CLAW extracts the LTE cell load, packet loss and bit-rate information based on the PHY statistics available locally on a smartphone. By exposing such fine-grained information, CLAW unleashes the transport-layer from data-driven bandwidth probing, and enables judicious rate control under channel dynamics and losses. CLAW improves mobile Web performance remarkably, in terms of both average latency and stability. We believe its PHY-informed design can benefit a broader range of mobile protocols and applications.

Acknowledgement

We appreciate the anonymous reviewers for their insightful comments. This research was supported in part by the NSF under Grant CNS-1343363, CNS-1350039, CNS-1506657 and CNS-1518728.

9. REFERENCES

[1] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan, "Reducing Web Latency: The Virtue of Gentle Aggression," in *ACM SIGCOMM*, 2013.

[2] J. Huang, F. Qian, Y. Guo, Y. Zhou, Q. Xu, Z. M. Mao, S. Sen, and O. Spatscheck, "An In-depth Study of LTE: Effect of Network Protocol and Application Behavior on Performance," in *ACM SIGCOMM*, 2013.

[3] J. Erman, V. Gopalakrishnan, R. Jana, and K. K. Ramakrishnan, "Towards a SPDY'Ier Mobile Web?" in *ACM CoNEXT*, 2013.

[4] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "How Speedy is SPDY?" in *USENIX NSDI*, 2014.

[5] K. Winstein, A. Sivaraman, and H. Balakrishnan, "Stochastic Forecasts Achieve High Throughput and Low Delay Over Cellular Networks," in *NSDI*, 2013.

[6] N. Becker, A. Rizk, and M. Fidler, "A Measurement Study on the Application-Level Performance of LTE," in *IFIP Networking Conference*, 2014.

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

[8] Y. Zhang, A. Arvidsson, M. Siekkinen, and G. Urvoy-Keller, "Understanding HTTP flow rates in cellular networks," in *IFIP Networking Conference*, 2014.

[9] K. I. Pedersen, T. E. Kolding, F. Frederiksen, I. Z. Kovacs, D. Laselva, and P. E. Mogensen, "An Overview of Downlink Radio Resource Management for UTRAN Long-term Evolution," *IEEE Communications Magazine*, 2009.

[10] X. Xie, X. Zhang, S. Kumar, and L. E. Li, "piStream: Physical Layer Informed Adaptive Video Streaming Over LTE," in *ACM MobiCom*, 2015.

[11] S. Kumar, E. Hamed, D. Katabi, and L. Erran Li, "LTE Radio Analytics Made Easy and Accessible," in *ACM SIGCOMM*, 2014.

[12] F. Lu, H. Du, A. Jain, G. M. Voelker, A. C. Snoeren, and A. Terzis, "CQIC: Revisiting Cross-Layer Congestion Control for Cellular Networks," in *ACM HotMobile*, 2015.

[13] Y. Zaki, T. Pötsch, J. Chen, L. Subramanian, and C. Görg, "Adaptive Congestion Control for Unpredictable Cellular Networks," in *ACM SIGCOMM*, 2015.

[14] A. Chakraborty, V. Navda, V. N. Padmanabhan, and R. Ramjee, "Coordinating Cellular Background Transfers Using Loadsense," in *ACM MobiCom*, 2013.

[15] J. Erman, A. Gerber, M. Hajiaghayi, D. Pei, S. Sen, and O. Spatscheck, "To Cache or Not to Cache: The 3G Case," *IEEE Internet Computing*, vol. 15, no. 2, 2011.

[16] IETF, "Hypertext Transfer Protocol Version 2 (HTTP/2)," https://tools.ietf.org/html/rfc7540/, 2015.

[17] S. Sundaresan, N. Feamster, R. Teixeira, and N. Magharei, "Measuring and Mitigating Web Performance Bottlenecks in Broadband Access Networks," in *Proc. of Internet Measurement Conference*, 2013.

[18] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie, “Why are web browsers slow on smartphones?” in *Proc. of the Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2011.

[19] N. Larson, D. Baltrunas, A. Kvalbein, A. Dhamdhere, k. claffy, and A. Elmokashfi, “Investigating Excessive Delays in Mobile Broadband Networks,” in *Proc. of the Workshop on All Things Cellular (AllThingsCellular)*, 2015.

[20] P. Sun, M. Yu, M. J. Freedman, and J. Rexford, “Identifying Performance Bottlenecks in CDNs Through TCP-level Monitoring,” in *Proc. ACM SIGCOMM Workshop on Measurements Up the Stack (W-MUST)*, 2011.

[21] D. Katabi, M. Handley, and C. Rohrs, “Congestion Control for High Bandwidth-Delay Product Networks,” *ACM SIGCOMM Computer Communication Review*, 2002.

[22] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, “Design and Implementation of a Routing Control Platform,” in *USENIX NSDI*, 2005.

[23] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, “PCC: Re-architecting Congestion Control for Consistent High Performance,” in *NSDI*, 2015.

[24] V. Srivastava and M. Motani, “Cross-layer Design: A Survey and the Road Ahead,” *IEEE Communication Magazine*, vol. 43, no. 12, 2005.

[25] B. Fu, Y. Xiao, H. J. Deng, and H. Zeng, “A Survey of Cross-Layer Designs in Wireless Networks,” *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, 2014.

[26] R. Kateja, N. Baranasuriya, V. Navda, and V. N. Padmanabhan, “DiversiFi: Robust Multi-Link Interactive Streaming,” in *ACM CoNEXT*. ACM, 2015.

[27] N. Baranasuriya, V. Navda, V. N. Padmanabhan, and S. Gilbert, “QProbe: Locating the Bottleneck in Cellular Communication,” in *CoNEXT*, 2015.

[28] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan, “Reducing Web Latency: The Virtue of Gentle Aggression,” in *ACM SIGCOMM*, 2013.

[29] 3GPP, “LTE: Evolved Universal Terrestrial Radio Access (E-UTRA); Physical channels and modulation,” *TS 36.211 version 12.3.0 Release 12*, 2014.

[30] N. Bui and J. Widmer, “OWL: A Reliable Online Watcher for LTE Control Channel Measurements,” in *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, 2016.

[31] 3GPP, “LTE: Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures,” *TS 36.213 version 12.4.0 Release 12*, 2015.

[32] Y. Li, C. Peng, Z. Yuan, J. Li, H. Deng, and T. Wang, “MobileInsight: Extracting and Analyzing Cellular Network Information on Smartphones,” in *ACM MobiCom*, 2016.

[33] “The Apache HTTP Server Project.” [Online]. Available: https://httpd.apache.org/

[34] IETF HTTP Working Group, “HTTP/2 Wiki.” [Online]. Available: https://github.com/http2/http2-spec/wiki/Ops

[35] Yasir Zaki, “Verus Source Code on GitHub.” [Online]. Available: https://github.com/yzaki/verus/

[36] Google, “Remote Debugging Android Devices.” [Online]. Available: https://developers.google.com/web/tools/chrome-devtools/debug/remote-debugging/remote-debugging?hl=en

[37] R. Netravali, A. Goyal, J. Mickens, and H. Balakrishnan, “Polaris: Faster Page Loads Using Fine-grained Dependency Tracking,” in *USENIX NSDI*, 2016.

[38] Y. Zhu, M. Halpern, and V. J. Reddi, “Event-Based Scheduling for Energy-Efficient qos (eqos) in Mobile Web Applications,” in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015.

[39] X. Li and Z. Bao, “Performance Characterization of Web Applications with HTML5 Enhancements,” in *Dependable, Autonomic and Secure Computing (DASC), 2014 IEEE 12th International Conference on*. IEEE, 2014.

[40] J. Gettys and K. Nichols, “Bufferbloat: Dark Buffers in the Internet,” *Communications of the ACM*, vol. 55, no. 1, 2012.

[41] M. Carbone and L. Rizzo, “Dummynet revisited,” in *ACM SIGCOMM Computer Communication Review*, 2010.

[42] Alexa. [Online]. Available: http://www.alexa.com/

[43] U. Hengartner, J. Bolliger, and T. Gross, “TCP Vegas Revisited,” in *INFOCOM 2000*, 2000.

[44] 3GPP, “LTE UE-Category.” [Online]. Available: http://www.3gpp.org/keywords-acronyms/1612-ue-category

[45] Qualcomm, “Snapdragon 800 Processor.” [Online]. Available: https://www.qualcomm.com/products/snapdragon/processors/800

[46] ——, “Snapdragon 808 Processor.” [Online]. Available: https://www.qualcomm.com/products/snapdragon/processors/808

[47] A. Jain, A. Terzis, N. Sprecher, S. Arunachalam, K. Smith, and G. Klas, “Requirements and reference architecture for Mobile Throughput Guidance Exposure,” 2015. [Online]. Available: https://tools.ietf.org/html/draft-flinck-mobile-throughput-guidance-03

[48] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, “A close examination of performance and power characteristics of 4g lte networks,” in *ACM MobiSys*, 2012.

[49] J. Nejati and A. Balasubramanian, “An In-Depth Study of Mobile Browser Performance,” in *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2016.

[50] Google, “Accelerated Mobile Pages (AMP),” 2016. [Online]. Available: https://www.ampproject.org