Accelerating Mobile Applications with Parallel High-bandwidth and Low-latency Channels

Sentosa, William; Chandrasekaran, Balakrishnan; Brighten Godfrey, P.; Hassanieh, Haitham; Maggs, Bruce; Singla, Ankit

published in
HotMobile 2021
2021

DOI (link to publisher)
10.1145/3446382.3448357

document version
Publisher's PDF, also known as Version of record

Accelerating Mobile Applications With Parallel High-bandwidth and Low-latency Channels

William Sentosa[1], Balakrishnan Chandrasekaran[2], P. Brighten Godfrey[1,3], Haitham Hassanieh[1], Bruce Maggs[4], Ankit Singla[5]
[1]University of Illinois at Urbana-Champaign, [2]VU Amsterdam and MPI-INF, [3]VMWare, [4]Duke University, Emerald Innovations, and MIT, [5]ETH Zürich

ABSTRACT

Interactive mobile applications like web browsing and gaming are known to benefit significantly from low latency networking, as applications communicate with cloud servers and other users' devices. Emerging mobile channel standards have not met these needs: general-purpose channels are greatly improving bandwidth but empirically offer little improvement for common latency-sensitive applications, and ultra-low-latency channels are targeted at only specific applications with very low bandwidth requirements.

We explore a different direction for wireless channel design: utilizing two channels – one high bandwidth, one low latency – simultaneously for general-purpose applications. With a focus on web browsing, we design fine-grained traffic steering heuristics that can be implemented in a shim layer of the host network stack, effectively exploiting the high bandwidth and low latency properties of both channels. In the special case of 5G's channels, our experiments show that even though URLLC offers just 0.2% of the bandwidth of eMBB, the use of both channels in parallel can reduce page load time by 26% to 59% compared to delivering traffic exclusively on eMBB. We believe this approach may benefit applications in addition to web browsing, may offer service providers incentives to deploy low latency channels, and suggests a direction for the design of future wireless channels.

CCS CONCEPTS

• **Networks** → **Mobile networks**; **Network architectures**.

KEYWORDS

5G, eMBB, URLLC, Traffic steering, Parallel channels

ACM Reference Format:
William Sentosa[1], Balakrishnan Chandrasekaran[2], P. Brighten Godfrey[1,3], Haitham Hassanieh[1], Bruce Maggs[4], Ankit Singla[5], [1]University of Illinois at Urbana-Champaign, [2]VU Amsterdam and MPI-INF, [3]VMWare, [4]Duke University, Emerald Innovations, and MIT, [5]ETH Zürich . 2021. Accelerating Mobile Applications With Parallel High-bandwidth and Low-latency Channels. In *The 22nd International Workshop on Mobile Computing Systems and Applications (HotMobile '21), February 24–26, 2021, Virtual, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3446382.3448357

HotMobile '21, February 24–26, 2021, Virtual, United Kingdom

ACM ISBN 978-1-4503-8323-3/21/02…$15.00
https://doi.org/10.1145/3446382.3448357

1 INTRODUCTION

Low latency is critical to interactive applications such as web browsing, virtual and augmented reality, and cloud gaming. For web applications, even an increase of 100 ms latency can result in as much as 1% revenue loss, as noted by Amazon [11]. The emerging VR/AR and cloud gaming applications also rely on the low latency link to deliver a seamless user experience. For instance, VR requires 20 ms or lower latency to avoid any simulator sickness [10].

Current mobile networks, serving general Internet applications such as web browsing and video streaming, have not yet delivered consistent low latency performance. This is fundamentally challenging due to the inherent tradeoff between latency and bandwidth [14]. One approach is to provide two separate channels (or services) – one optimizing for bandwidth, the other optimizing for latency – with different types of user applications assigned to them. For example, 5G NR follows this pattern with its enhanced mobile broadband (eMBB) and ultra-reliable and low-latency communication (URLLC) channels. eMBB, which serves general-purpose Internet use, is heavily focused on delivering gigabit bandwidth. This channel will be useful for streaming media, but offers little improvement for latency-sensitive applications, such as web browsing. Experimentally, web page load time in existing 5G deployments, even in close-to-ideal circumstances (stationary device and little utilization), is similar to 4G for pages of size < 3 MB and about 19% faster than 4G for sites > 3 MB [24]; this discrepancy is due to 5G eMBB having 28 ms or larger latency, broadly similar to 4G.

Meanwhile, 5G URLLC promises an exciting capability of very low latency, in the range of 2 to 10 ms [3], but compromises severely on bandwidth, making it unsuitable for common mobile applications. Our experiments emulating web browsing (the most widely used mobile application [32], and far from the most bandwidth-intensive application) over URLLC with 2 Mbps bandwidth show web page load times would be 5.87× higher than with eMBB.

As the latency-bandwidth tradeoff is fundamental, we expect this separation between channels is likely to persist; 6G, for example, is also expected to include a low latency channel [38]. But we believe the availability of both a high bandwidth channel (HBC) and a low latency channel (LLC) offers an opportunity beyond simple static assignment of an application to a single channel. Our hypothesis is that *by using high bandwidth and low latency channels in parallel on mobile devices, significant performance and user experience*

improvements are possible for latency-sensitive applications. Here, we explore this hypothesis for the case of web browsing.

Designing a way of mapping an application's traffic to HBC and LLC is difficult since we have to use LLC's bandwidth very selectively. Indeed, the main deployed mechanism to combine multiple channels, MPTCP [30], assumes two interfaces that are each of significant bandwidth, with the goal of aggregating that bandwidth or supporting failover. LLC's bandwidth, however, is a rounding error compared to HBC's. Other works – particularly Socket Intents [31] and TAPS [27] – exploit multi-access connectivity through application-level input, which we prefer to avoid to ease deployment and expand relevance to other applications in the future; therefore we expect new mechanisms are necessary.

To solve these problems, one approach would be to steer traffic at the granularity of flows or web objects (e.g., large vs. small). To be as selective as possible in our use of LLC, we design a fine-grained packet-level steering scheme. The scheme uses simple heuristics to identify packets that are more likely to help the application if they are accelerated, while using a very limited amount of state to minimize the out-of-order delivery that can result from accelerating only a subset of packets in a flow. Since this steering scheme does not solicit extra information from applications, we can present a virtual network interface to the application, and transparently steer each packet sent to that interface over HBC or LLC.

To evaluate our design with a concrete scenario, we leverage 5G's eMBB and URLLC as our HBC and LLC. We utilize an emulation environment that includes a typical mobile web browsing scenario: a client with two network channels emulating characteristics of eMBB and URLLC radio access networks and the cellular core, as well as CDN edge, parent, and origin servers. We populate this setup with landing page snapshots of 53 popular web sites and test various assumptions about eMBB and URLLC performance. We evaluate several object and packet steering schemes, and prior approaches such as MPTCP [1] and ASAP [21]. Our findings conclude that packet-granularity schemes, which require little to no per-connection state and no application knowledge, provide the best performance. For the best steering scheme, even with a modest bandwidth of 2 Mbps allocated to URLLC, we can reduce mean page load time (PLT) by 26% compared to exclusively using eMBB. If eMBB's latency fluctuates, increasing from 30 ms to 150 ms (which is typical for mmWave channels [15, 19]), combining eMBB and URLLC decreases mean PLT by 59%.

Finally, we discuss deployment strategies, challenges, and future opportunities. Although we have focused on web browsing in this short paper, we believe our basic techniques can apply to a wide variety of latency-sensitive applications, and open new opportunities for app developers and cellular providers.

2 BACKGROUND & RELATED WORK

Web browsing traffic: Web browsing traffic is characterized by its often short and bursty flows, in which latency matters more than bandwidth. A web page easily contains tens to hundreds of relatively small-sized objects distributed across multiple servers and domains. A study across Alexa Top 200 pages found that the median number of objects in a page is 30, while the median object size is 17 KB [36]. This translates to many HTTP request-and-response interactions across many short flows. Page load time (PLT) is thus typically dominated by DNS lookup, connection establishment, and TCP convergence time – which require little throughput but are highly dependent on RTT. Increasing TCP throughput beyond $\approx$ 16 Mbps offers little improvement in PLT [33].

Channels in 5G: The 5G NR (New Radio) standard provides differentiated services to support diverse applications. (1) *Enhanced mobile broadband* (eMBB) is an HBC that serves standard mobile Internet applications. It is considered to be an upgraded 4G mobile broadband with higher data rates. A key enabling technology is the use of mmWave bands which offer high bandwidth. A recent measurement study on commercial mmWave 5G networks in the US shows TCP throughput of up to 2 Gbps for download and 60 Mbps for upload, with a mean RTT of 28 ms measured between the client and the first-hop edge server right outside the cellular network core [24].[1] (2) *Ultra-reliable low-latency communication* (URLLC), as an LLC, is intended for mission-critical and specialized latency-sensitive applications such as self-driving cars, factory automation, and remote surgery. It focuses on providing highly reliable, very low latency communication at the cost of limited throughput. While URLLC is yet to be deployed, the standard is finalized and specifies a 0.5 ms air latency (1 ms RTT) between the client and the Radio Access Network (RAN) with 99.999% reliability for small packets (e.g. 32 to 250 bytes) [6]. It also specifies a target end-to-end latency (from a client to a destination typically right outside the cellular network core) between 2 to 10 ms with throughput in the range of 4 Mbps [3]. URLLC is expected to guarantee such performance through dedicated network resources achieved via network slicing [3].

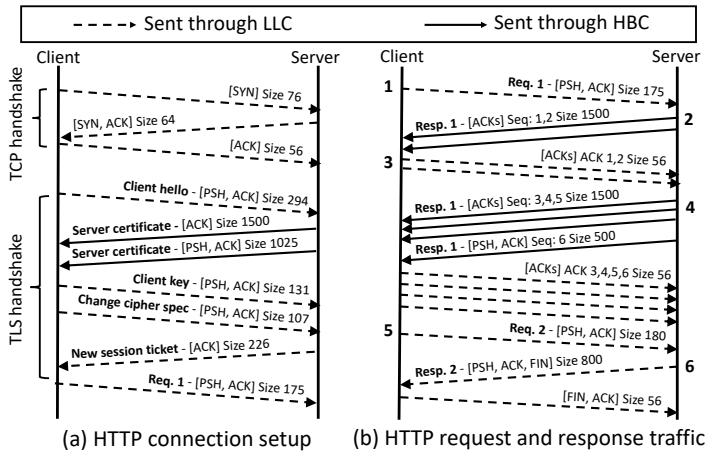
Leveraging multiple connections: MPTCP [30] provides an end-to-end transport protocol that supports multiple interfaces. Socket Intents [31] and Intentional networking [20] both expose custom API to application and offer OS-level support for managing multiple interfaces. Both of them regulate application traffic based on the application-specific information. These efforts are complimentary to ours since we focus on mapping the application traffic to HBC and LLC. Perhaps, works most relevant to ours are IANS [12] and ASAP [21]. IANS leverages Socket Intents to improve web page load. Unlike our approach (i.e., packet steering schemes), however, they require application-specific information (i.e., object size) to determine which interface to use and they work at the object level. We compare ASAP with our scheme in §4.2, and found that our scheme performs better due to its finer-grained decision.

3 TRAFFIC SPLITTING SCHEMES

In general, a scheme to split traffic between HBC and LLC needs to judiciously steer only a small portion of traffic through LLC to avoid overloading it (which would congest queues, negating latency benefits); and that small portion should yield the most benefit to application-level performance.

We investigate heuristics for web browsing in *object-level* and *packet-level* which employ different tradeoffs. Object-level splitting

[1] These measurements, however, were performed in conditions favorable to mmWave with line of sight, no mobility, and few clients. eMBB latency is expected to be higher as the number of users increases and as users move, especially for mmWave since the user or other obstacles can easily block the beam, leading to unreliable and inconsistent performance [2, 23, 24].



1. Client sends an HTTP GET request to the server. The size varies by its header and content, but it is typically small enough to fit into one packet.
2. Server sends the HTTP response data in multiple packets since the data size is more than MTU (1500 bytes). For the purpose of illustration, we set *init_cwnd*=2, and thus it can only send two packets. It needs to wait for an ACK from the client before sending more.
3. Client acknowledges data packets from Server.
4. Server sends another stream of packets, doubling from the first stream (no 2). The response data ends, and the last packet is *not* an MTU-sized packet.
5. Client reuses the connection to send another HTTP request.
6. The second response fits into a single packet. The server also sends FIN packet to tear down the connection.

Figure 1: HTTP traffic delivery on TCP. It also shows per-packet mapping done by the packet steering scheme (*pkt-state*).

may benefit from application-level knowledge about web objects, but requires application input. Packet-level splitting can benefit from finer-grained decisions, but needs to deal with challenges of out-of-order delivery (OOD) if packets within a flow are sent on different channels. We consider several natural schemes in each class, and design a new packet-level scheme that minimizes OOD.

Object-level splitting: A web page is comprised of many objects that vary in size. Inspired by prior works [12, 34], the simplest splitting heuristic is to request objects smaller than some threshold over LLC, as their load times are dominated by latency. All other objects use HBC. Note that this approach requires the application to specify object size, or to make the splitting choice itself. Our evaluation (with URLLC and eMBB) tests this *obj-size* scheme using two different thresholds of 3 KB and 6 KB. (The reasoning is that URLLC supports a maximum rate of ≈250 bytes per ms, and its RTT difference with eMBB is ≈25 ms. Therefore, objects of size >6.25 KB are likely delivered faster on eMBB.)

Another approach is to leverage application-level priority of web objects. Web pages have complex dependency structures, and certain objects can be on the critical path for web page loading. We evaluated a scheme (*obj-priority*) by utilizing Chrome priority assignment [17] to only fetch the *highest* priority object via LLC. Note that this scheme may not be a perfect fit either, because high priority objects could be large (HTML pages, which tend to have the highest priority, can be much larger than 6.25 KB) and will load faster via HBC.

Packet-level splitting: A simple packet-level heuristic is to send all client-to-server traffic over LLC and all server-to-client traffic over HBC; we call this scheme *pkt-uplink*. The intuition is that client requests are generally small. This scheme might not perform well, however, if the client sends a large amount of data to the server (e.g., using HTTP POST), and misses acceleration opportunities from server to client.

To devise a better heuristic, we investigate the traffic that arises when loading a typical web page. Before loading a page, the client exchanges small UDP packets (<250 bytes) with the nameserver to perform DNS lookup; delivering them on URLLC can accelerate the lookup process. The client continues by exchanging TCP packets with the web server as shown in Figure 1. Packets in the TCP flow can be categorized as *control packets* and *data packets*. *Control packets* are packets only intended to manage TCP operations (e.g., SYN and ACK packets), and do not carry any payload. These packets are tiny, and accelerating them will benefit the entire flow. For instance, accelerating SYN packets allows faster connection establishment while accelerating the client's ACK packets reduces flow RTT, which is crucial because RTT largely determines web object download times for small objects. Thus, we include a scheme (*pkt-control*) that only offloads control packets to LLC in our evaluation.

Data packets are packets that carry a payload. Mapping *data packets* to LLC can quickly consume the bandwidth, yet each packet does not provide equal benefit from acceleration. In Figure 1b, accelerating a single packet in the *second* HTTP response will complete the response, while accelerating a mere single in to the *first* HTTP response will cause head of line blocking. We observed that when a data cannot fit into one packet, it will be fragmented into multiple packets, with most of them being MTU-sized (see HTTP response 1 in Figure 1b). Thus, MTU-sized packets are likely belongs to a larger transfer, and thus are *not* worth accelerating.

However, not all non-MTU-sized packets are worth accelerating. For instance, the last (tail) packet of the HTTP response (Seq 6 in Figure 1b) is most likely less than MTU, and accelerating that will result in out-of-order delivery (OOD) that may confuse application. We can identify tail data packet with a simple *per-connection state* to remember its one preceding packet; if its preceding packet is MTU-sized, this packet is more likely to be the tail packet.

Guided by all these observations, we devise a simple packet steering scheme (*pkt-state*) that can determine whether a packet should be sent through LLC. It stores state for each connection to remember the size of the last packet delivered, and uses only packet size to make decisions. Figure 1 shows the end-result of packet-level mapping of the *pkt-state* for HTTP traffic. Leveraging this scheme, all non-MTU-sized packets except the tail data packets will be delivered through LLC, including DNS packets, control packets, and small data packets. Even though we were inspired by the HTTP traffic pattern in designing this scheme, we believe that this approach can be beneficial to other application traffic as well since our approach is application-agnostic.

Algorithm 1: PKT-State

Result: Send a packet to either HBC or LLC
if *size(pkt) < MTU and size(prev_pkt) < MTU* **then**
  send(pkt, LLC); /* control or small data pkt */
else
  send(pkt, HBC); /* tail or MTU-sized pkt */
end

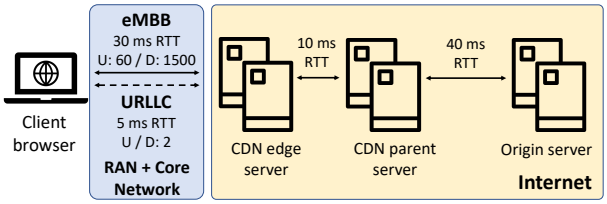


Figure 2: Baseline emulation setup

4 EVALUATION

We evaluate performance with emulated 5G eMBB and URLLC, and with parameters representing possible future channels.

4.1 Emulation and experimental setup

Our emulation setup (Figure 2) reflects the dependence of web applications on content delivery networks (CDNs). CDNs comprise a hierarchy of caches [9]. Here, the client's request first reaches an edge server. In case of a cache miss it is forwarded to a parent server, and in case of a second cache miss it is forwarded to the origin server storing the content. Request latency is therefore variable.

The client has two parallel paths (along eMBB and URLLC) to the edge server. The bandwidth and RTT of the eMBB path (1500 Mbps + 30 ms) is based on recent 5G measurements [24], and for URLLC (2 Mbps + 5 ms) on its sample use cases [3]. This base eMBB configuration is based on perfect line of sight with no mobility. For the CDN latency, we assume that both the edge and parent servers are located in the US midwest while the origin server is located on the US west coast. Based on publicly available network latency information [5], we use 10 ms RTT from edge to parent and 40 ms RTT from parent to origin.

Our experiments use a machine running Ubuntu 16.04 LTS with 32 GB RAM, 512 GB SSD, and an Intel Core i7-9700 CPU. The web server runs on Apache 2.4.18, while the client uses Chromium v81. The experiments use HTTP 1.1, leaving an evaluation of HTTP/2 and QUIC to future work.

We use Mahimahi [25] to record and replay web pages. We modified Mahimahi to include the CDN hierarchy, our packet steering techniques in Mahimahi's packet scheduling modules, and an HTTP proxy to enable object-level steering.

To realistically emulate cache misses and hits in the CDN, our evaluation uses web pages for which we can get this information from HTTP headers. HTTP pragma headers, used by CDNs for debugging, can be used to glean this information [29]. We started with a random sample of top-ranked Alexa pages and found 53 that provide the pragma headers. We recorded and locally replayed these sites' landing pages, whose mean size is 2.13 MB and whose 95th percentile size is 6.13 MB. We replayed each page load five times and reported its median PLT (based on the onLoad event [26]). We used cold DNS and browser caches for all experiments.

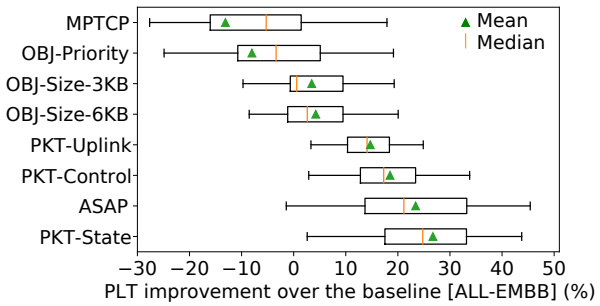


Figure 3: Comparisons between object-level and packet-level steering. The baseline has a mean PLT of 1635 ms.

4.2 Evaluation of multiple schemes

We evaluated three object-level and three packet-level steering schemes (§3), and compared them with a baseline where the client only utilized eMBB (*all-embb*), and with Multipath TCP (MPTCP) [1] and ASAP [21]. MPTCP spreads traffic across multiple interfaces; we evaluated the default Linux kernel implementation. ASAP was designed to accelerate PLT in satellite networks by offloading some traffic to 3G/4G (which had lower latency but higher cost per byte than satellites, in [21]). Figure 3 and Table 1 summarize the results.

Object-level splitting performs poorly. Obj-Priority is often worse than the baseline since it sends high priority objects to the low-latency channel, but these can be large. The size-based heuristics do not improve much compared to the baseline – only about 25% of web pages have > 10% lower PLT – and they are sometimes *worse*. This is because many small objects (like icons) do not affect PLT much. Some pages perform slightly worse because of resource contention on URLLC when fetching multiple small objects in parallel.

Packet-level splitting, in contrast, performs consistently well, especially *pkt-state*, which has 1243 ms mean PLT (a 26% improvement) while only offloading 12.6% of bytes to URLLC. Interestingly, *pkt-control* improves mean PLT by 18.5% by only offloading control packets comprising 3.1% of bytes. These improvements stem from the finer granularity of splitting, such that *every* web object experiences faster download, ultimately improving PLT for *all* pages. Unsurprisingly, the smaller web pages tend to improve more, as their PLT is more dominated by latency than bandwidth.

Understanding *pkt-state* gains: To understand the network-related gain of *pkt-state*, we use *curl* to download the HTML document of *amazon.com* within our emulation setup. This object is cached at the edge, so no CDN delay is incurred. Table 2 shows that *pkt-state* performs better than not only *all-embb*, but also *all-urllc*. This is because *pkt-state* offloads (1) DNS packets that reduce DNS lookup time, (2) *control packets* that reduce connection (SYN packets) and object transfer time (Client's ACK packets), and (3) small *data packets* that accelerate small data transfer such as TLS client key transfer or HTTP request data. Meanwhile, it benefits from the higher bandwidth than *all-urllc* which contributes to a better transfer time.

Comparison with existing works: MPTCP generally fails to perform better than always using a single eMBB path. Vanilla MPTCP is known to perform sub-optimally under heterogeneous paths due to head-of-line blocking [13] and its policy to lean towards load balancing among multiple paths. A superior MPTCP scheduler may

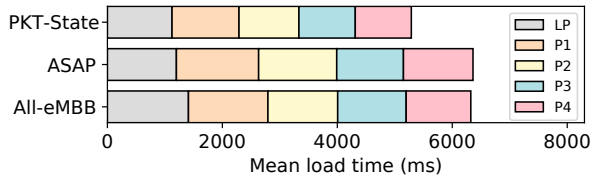


Figure 4: Mean load time across different pages: the landing page (LP) and four internal pages (P1, P2, P3, P4)

Steering scheme	Mean bytes per page	% bytes		Avg rate (Mbps)	
		Mean	90th	Mean	90th
MPTCP [1]	476.3 KB	22.1%	41.6%	U: 0.15 D: 0.68	U: 0.24 D: 1.02
OBJ-Priority	259.6 KB	12.8%	29%	U: 0.10 D: 1.13	U: 0.14 D: 1.75
OBJ-Size-3KB	118.1 KB	5.5%	11%	U: 0.21 D: 0.45	U: 0.42 D: 0.82
OBJ-Size-6KB	242.8 KB	10.7%	20.3%	U: 0.28 D: 0.73	U: 0.46 D: 1.32
PKT-Uplink	186.7 KB	8.6%	13.5%	U: 0.38 D: 0	U: 0.66 D: 0
PKT-Control	62.8 KB	3.1%	4.6%	U: 0.1 D: 0.04	U: 0.17 D: 0.07
ASAP [21]	257 KB	12.3%	20.3%	U: 0.85 D: 0.50	U: 1.26 D: 0.77
PKT-State	260.5 KB	12.6%	20.4%	U: 0.87 D: 0.52	U: 1.28 D: 0.81

Table 1: Traffic sent over URLLC by each scheme, in bytes per page, percent of total bytes of page load, and mean upload (U) and download (D) rate.

achieve better results, and could provide a deployment path for parts of our work (§5).

ASAP performs similar to *pkt-state* with 23% mean improvement. ASAP splits packets based on its corresponding HTTP phase; it accelerates DNS, connection (TLS + SSL) handshake, and HTTP request traffic, while leaving HTTP responses for eMBB. This assumes that HTTP requests are small, which is mostly true when a browser loads a single landing page, but does not hold in general. A simple example is when a user uploads a photo, but we also found that ASAP encounters problems as the user browses deeper into the site.

Specifically, we performed a smaller-scale experiment to simulate an interactive browser session. We picked 21 web sites from our corpus that have "hamburger" menu buttons, and for each site we performed an automated click through four different menus in the same browser session. ASAP performs worse as subsequent pages are loaded (Figure 4). This happens because as the user clicks through internal links, more data is pushed to the server, resulting in larger HTTP request traffic, as shown by the increased request size on the 99th percentile – 1.92 KB per page for landing pages compared to 2.9 KB when visiting four additional pages.

In summary, ASAP has several drawbacks: slightly worse mean and variance PLT for the landing page, noticeably worse PLT for subsequent pages, and is more tied to HTTP.

Perf. metric	*All-eMBB*	*All-URLLC*	*PKT-State*
DNS lookup	60 ms	12 ms	12 ms
TCP connect	32 ms	6 ms	6 ms
TLS connect	180 ms	94 ms	92 ms
Object transfer	92 ms	427 ms	54 ms
Total loading time	399 ms	579 ms	183 ms
Avg. download rate	2.2 Mbps	1.5 Mbps	4.8 Mbps

Table 2: A deeper look at the *pkt-state* performance when fetching the landing page of amazon.com (110 KB).

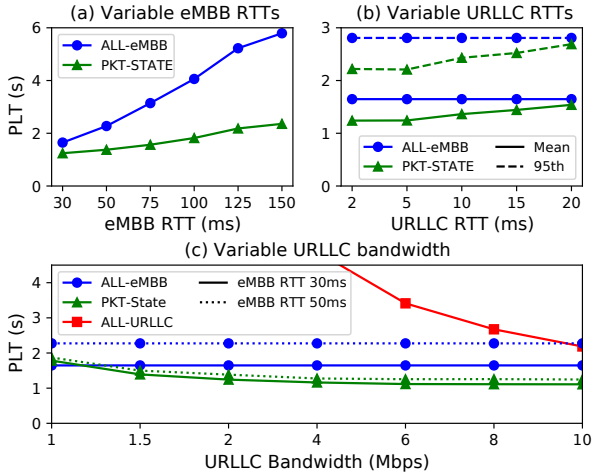


Figure 5: Varying eMBB and URLLC network conditions, conducted using the baseline setup (Figure 2) with a single varying parameter, except (c) where we change two parameters.

4.3 Varying network performance

We evaluated the best heuristic (*pkt-state*) under different LLC and HBC network characteristics.

Varying eMBB network RTTs: RTT inflation is common in mobile networks due to bufferbloat [18, 22] and physical properties, especially under movement. We evaluated inflated RTT on the HBC, and found *pkt-state* provides even larger improvements (Figure 5a). *pkt-state* has 59% (≈ 2.4×) lower PLT than *all-embb* when the eMBB RTT is 150 ms, which is possible [24] when the channel is heavily utilized. Since *pkt-state* uses eMBB primarily for bandwidth-intensive traffic, it is extremely robust to worsened RTT on the main channel.

Varying URLLC network RTTs: Although URLLC is expected to deliver consistent low latency, we evaluate latency inflation on URLLC to reflect scenarios where web traffic is de-prioritized in favor of critical traffic. *pkt-state* is still superior with URLLC latency increasing up to 20 ms and eMBB held at 30 ms (Figure 5b). To prevent worse performance, the client could periodically probe the LLC and if its latency is too high, fall back to using the HBC.

Varying URLLC bandwidth: *pkt-state* requires little bandwidth: its PLT flattens for more than 2 Mbps (Figure 5c). This is expected since its average data rate is only 0.87 Mbps up and 0.52 Mbps down (Table 1). The figure also shows we cannot entirely rely on URLLC (*all-urllc*) as it needs more than 10 Mbps to approach the baseline.

5 DEPLOYMENT IN 5G

We discuss the packet steering deployment in 5G eMBB and URLLC and its challenges. They are all parts of the future work and require further evaluation; however, we outline our expectations.

Packet steering implementation: In the deployment, two specific changes are required: packet steering in the user device operating system, and in an existing proxy in the network provider. The steering should be performed in the network-layer, and hence we do not expect to modify both eMBB and URLLC link- and physical-layer design. On the client-side, the operating system needs to steer packets over eMBB and URLLC interfaces. One approach is to present a virtual interface that ties both interfaces and steers packets via the custom bridging [37]. Applications wishing to utilize parallel channels may use the provided virtual interface. Another way is to leverage MPTCP and introduce a new scheduling module. The former approach may be preferred because MTPCP also requires MPTCP-compliant web servers, and it cannot steer non-MPTCP traffic. We do not need any or minimal (such as to use the virtual interface) application-level change as our approach is transparent to the application.

Since traffic steering should be performed in both directions, network providers need to steer packets originating from the Internet and merge URLLC and eMBB traffic from the client. This change should be minimal since they can directly leverage an existing proxy in the core network, such as the packet data network gateway (P-GW). This approach ensures transparency of the packet steering to the server.

URLLC scalability: The number of users that can send general traffic to URLLC is an important matter which deserves to be evaluated quantitatively in the future. At the time of the writing, URLLC is not yet deployed in public. However, based on the white paper [4], URLLC is targeted to support a relatively high connection density with modest per-user bandwidth. For instance, one of the URLLC use cases (discrete automation) requires a user-experienced data rate of 10 Mbps, traffic density of 1 Tbps/km2, connection density of 100.000/km2, and max end-to-end latency of 10ms. We expect URLLC to reserve 2 Mbps maximum bandwidth per user for general application traffic, which is still reasonable based on others' proposed use cases for URLLC, even in a dense urban area.

Disrupting URLLC native traffic: URLLC is primarily built to serve latency-sensitive critical applications. To ensure we do not compromise the performance of these applications, we only use a small amount of the URLLC capacity. In particular, the 90^{th} percentile average data rate is only 1.28 Mbps (Table 1). Moreover, the network operator can limit the per-user bandwidth and even choose to deprioritize non-critical packets as our approach does not require 99.999% reliability and is resilient to the slight increase in URLLC latency (§4.3).

Resource contention among applications: Multiple applications inside a user equipment may compete to use URLLC. We can regulate them using prioritization. One simple approach is to prioritize applications running in the foreground since mobile phone users are typically single-tasking.

Incentives for operators: While URLLC targets critical applications, it is up to the network providers to open URLLC for general mobile applications like web browsing. This is possible as 5G chipsets are typically designed to support multiple bands including the sub-6GHz bands for URLLC [3]. Expanding the applications of URLLC can encourage providers to foster a faster and broader deployment of URLLC as it brings a smoother experience to their major customers – mobile phone users; especially as the current market for URLLC applications like self-driving cars and remote surgery is still in its infancy.

6 RESEARCH OPPORTUNITIES

The use of LLC and HBC opens up new research opportunities.

Other applications: In addition to web browsing, our approach can benefit many *mobile applications* which typically employ HTTP but are different in terms of page contents. Their traffic is generally smaller since it is characterized by many JSON requests [35] and hence, it should benefit more from the improved latency compared to bandwidth. LLC and HBC combination can also properly support applications from different domains that require high-bandwidth and low-latency; something that cannot be satisfied by utilizing a mere single channel. For instance, cloud gaming, which allows users to play games from remote servers, requires high bandwidth to stream the content and low latency to remain responsive to user input. Since these applications can be vastly different than web browsing, a superior steering scheme may exist. We plan to analyze them further to determine an effective way of leveraging LLC and HBC.

Beyond mobile networks: Our insights may apply to other LLC and HBC combinations with analogous bandwidth and latency trade-offs. Examples include quality of service (QoS) differentiation providing separate latency- and bandwidth-optimized services [7, 28]; and routing traffic among multiple ISPs where one is more expensive but provides better latency, as may happen with very low Earth orbit satellite-based [16] or specialty [8] ISPs. To achieve the optimum cost to performance ratio, we can route only the latency-sensitive traffic to the low-latency ISP.

Future wireless design: The 5G URLLC is only equipped with limited user bandwidth, and hence it is not suitable to serve general application traffic. The bandwidth is severely compromised because it needs to provide *both* low latency and very high reliability (99.999%). However, general applications do not need the almost-perfect reliability that URLLC guarantees. Future wireless networks (such as 6G) may reconsider this trade-off and provide a low-latency channel with somewhat greater bandwidth and somewhat lower reliability.

7 CONCLUSION

We present an idea to utilize high-bandwidth and low-latency channels in parallel to improve mobile applications. In this early work, we show the way web browsing performance can be improved by carefully steering traffic over 5G eMBB and URLLC and exploiting the channels' massive bandwidth and low latency properties.

ACKNOWLEDGMENTS

We thank the anonymous reviewers and our shepherd Wenjun Hu for their comments. This work was supported by the National Science Foundation under Grant No. 1763841.

REFERENCES

[1] 2018. Multipath TCP in the Linux Kernel v0.94. http://www.multipath-tcp.org. [Last accessed on June 16, 2020].

[2] 2018. MWC: Are Your 5 Fingers Blocking Your 5G? https://www.eetimes.com/mwc-are-your-5-fingers-blocking-your-5g/. [Last accessed on June 24, 2020.

[3] 2019. 3GPP TR 38.824 Release 16. https://www.3gpp.org/release-16. [Last accessed on June 16, 2020].

[4] 2019. 3GPP TS 22.261 version 15.7.0 Release 15. https://www.etsi.org/deliver/etsi_TS/122200_122299/122261/15.07.00_60/ts_122261v150700p.pdf. [Last accessed on January 20, 2021].

[5] 2020. AS7029 Windstream Communication Network Latency. https://ipnetwork.windstream.net. [Last accessed on June 8, 2020].

[6] 3rd Generation Partnership Project. 2017. *Study on Scenarios and Requirements for Next Generation Access Technologies*. Technical Report.

[7] Jozef Babiarz, Kwok Chan, and Fred Baker. 2006. Configuration guidelines for DiffServ service classes. *Network Working Group* (2006).

[8] Debopam Bhattacherjee, Sangeetha Abdu Jyothi, Ilker Nadi Bozkurt, Muhammad Tirmazi, Waqar Aqeel, Anthony Aguirre, Balakrishnan Chandrasekaran, P Godfrey, Gregory P Laughlin, Bruce M Maggs, et al. 2018. cISP: A Speed-of-Light Internet Service Provider. *arXiv preprint arXiv:1809.10897* (2018).

[9] Anawat Chankhunthod, Peter B Danzig, Chuck Neerdaels, Michael F Schwartz, and Kurt J Worrell. 1996. A Hierarchical Internet Object Cache.. In *USENIX Annual Technical Conference*. 153–164.

[10] Eduardo Cuervo. 2017. Beyond reality: Head-mounted displays for mobile systems researchers. *GetMobile: Mobile Computing and Communications* 21, 2 (2017), 9–15.

[11] Yoav Einav. 2019. *Amazon Found Every 100ms of Latency Cost them 1% in Sales*. Retrieved May 24, 2020 from https://www.gigaspaces.com/blog/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/

[12] Theresa Enghardt, Philipp S Tiesel, Thomas Zinner, and Anja Feldmann. 2019. Informed Access Network Selection: The Benefits of Socket Intents for Web Performance. In *2019 15th International Conference on Network and Service Management (CNSM)*. IEEE, 1–9.

[13] Simone Ferlin, Özgü Alay, Olivier Mehani, and Roksana Boreli. 2016. BLEST: Blocking estimation-based MPTCP scheduler for heterogeneous networks. In *2016 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE, 431–439.

[14] A El Gamal, James Mammen, Balaji Prabhakar, and Devavrat Shah. 2004. Throughput-delay trade-off in wireless networks. In *IEEE INFOCOM 2004*, Vol. 1. IEEE.

[15] M. Giordani, M. Polese, A. Roy, D. Castor, and M. Zorzi. 2019. A Tutorial on Beam Management for 3GPP NR at mmWave Frequencies. *IEEE Communications Surveys Tutorials* 21, 1 (2019), 173–196.

[16] Giacomo Giuliari, Tobias Klenze, Markus Legner, David Basin, Adrian Perrig, and Ankit Singla. 2020. Internet backbones in space. *ACM SIGCOMM Computer Communication Review* 50, 1 (2020), 25–37.

[17] Sergio Gomes. 2020. Resource Prioritization – Getting the Browser to Help You. https://developers.google.com/web/fundamentals/performance/resource-prioritization. [Last accessed on June 12, 2020].

[18] Yihua Guo, Feng Qian, Qi Alfred Chen, Zhuoqing Morley Mao, and Subhabrata Sen. 2016. Understanding on-device bufferbloat for cellular upload. In *Proceedings of the 2016 Internet Measurement Conference*. 303–317.

[19] Haitham Hassanieh, Omid Abari, Michael Rodriguez, Mohammed Abdelghany, Dina Katabi, and Piotr Indyk. 2018. Fast Millimeter Wave Beam Alignment. In *SIGCOMM'18*.

[20] Brett D Higgins, Azarias Reda, Timur Alperovich, Jason Flinn, Thomas J Giuli, Brian Noble, and David Watson. 2010. Intentional networking: opportunistic exploitation of mobile network diversity. In *Proceedings of the sixteenth annual international conference on Mobile computing and networking*. 73–84.

[21] Se Gi Hong and Chi-Jiun Su. 2015. ASAP: fast, controllable, and deployable multiple networking system for satellite networks. In *2015 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 1–7.

[22] Haiqing Jiang, Yaogong Wang, Kyunghan Lee, and Injong Rhee. 2012. Tackling bufferbloat in 3G/4G networks. In *Proceedings of the 2012 Internet Measurement Conference*. 329–342.

[23] Adrian Loch, Irene Tejado, and Joerg Widmer. 2016. Potholes Ahead: Impact of Transient Link Blockage on Beam Steering in Practical mm-Wave Systems. In *The 22nd European Wireless Conference*.

[24] Arvind Narayanan, Eman Ramadan, Jason Carpenter, Qingxu Liu, Yu Liu, Feng Qian, and Zhi-Li Zhang. 2020. A First Look at Commercial 5G Performance on Smartphones. In *Proceedings of The Web Conference 2020*. 894–905.

[25] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. 2015. Mahimahi: Accurate record-and-replay for HTTP. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. 417–429.

[26] Jan Odvarko. 2007. *HAR 1.2 Spec*. Retrieved June 8, 2020 from http://www.softwareishard.com/blog/har-12-spec/

[27] Tommy Pauly, Brian Trammell, Anna Brunstrom, Gorry Fairhurst, Colin Perkins, Philipp S Tiesel, and Christopher A Wood. 2018. An architecture for transport services. *Internet-Draft draft-ietf-taps-arch-00, IETF* (2018).

[28] Maxim Podlesny and Sergey Gorinsky. 2008. RD network services: differentiation through performance incentives. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*. 255–266.

[29] Shankaranarayanan Puzhavakath Narayanan, Yun Seong Nam, Ashiwan Sivakumar, Balakrishnan Chandrasekaran, Bruce Maggs, and Sanjay Rao. 2016. Reducing latency through page-aware management of web objects by content delivery networks. *ACM SIGMETRICS Performance Evaluation Review* 44, 1 (2016), 89–100.

[30] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. 2012. How hard can it be? designing and implementing a deployable multipath TCP. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. 399–412.

[31] Philipp S Schmidt, Theresa Enghardt, Ramin Khalili, and Anja Feldmann. 2013. Socket intents: Leveraging application awareness for multi-access connectivity. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. 295–300.

[32] M Zubair Shafiq, Lusheng Ji, Alex X Liu, Jeffrey Pang, and Jia Wang. 2012. Characterizing geospatial dynamics of application usage in a 3G cellular data network. In *2012 Proceedings IEEE INFOCOM*. IEEE, 1341–1349.

[33] Srikanth Sundaresan, Nick Feamster, Renata Teixeira, and Nazanin Magharei. 2013. Measuring and mitigating web performance bottlenecks in broadband access networks. In *Proceedings of the 2013 conference on Internet measurement conference*. 213–226.

[34] Philipp S Tiesel, Theresa Enghardt, Mirko Palmer, and Anja Feldmann. 2018. Socket intents: Os support for using multiple access networks and its benefits for web browsing. *arXiv preprint arXiv:1804.08484* (2018).

[35] Santiago Vargas, Utkarsh Goel, Moritz Steiner, and Aruna Balasubramanian. 2019. Characterizing JSON Traffic Patterns on a CDN. In *Proceedings of the Internet Measurement Conference*. 195–201.

[36] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. 2014. How Speedy is SPDY?. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. 387–399.

[37] Kok-Kiong Yap, Te-Yuan Huang, Yiannis Yiakoumis, Sandeep Chinchali, Nick McKeown, and Sachin Katti. 2013. Scheduling packets over multiple interfaces while respecting user preferences. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. 109–120.

[38] Baiqing Zong, Chen Fan, Xiyu Wang, Xiangyang Duan, Baojie Wang, and Jianwei Wang. 2019. 6G technologies: Key drivers, core requirements, system architectures, and enabling technologies. *IEEE Vehicular Technology Magazine* 14, 3 (2019), 18–27.