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

William Sentosa, *University of Illinois Urbana-Champaign;*
Balakrishnan Chandrasekaran, *Vrije Universiteit Amsterdam;*
P. Brighten Godfrey, *University of Illinois Urbana-Champaign and VMware;*
Haitham Hassanieh, *EPFL;* Bruce Maggs, *Duke University and Emerald Innovations*

https://www.usenix.org/conference/nsdi23/presentation/sentosa

This paper is included in the
Proceedings of the 20th USENIX Symposium on
Networked Systems Design and Implementation.

April 17–19, 2023 • Boston, MA, USA

978-1-939133-33-5

Open access to the Proceedings of the
20th USENIX Symposium on Networked
Systems Design and Implementation
is sponsored by

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

William Sentosa[•], *Balakrishnan Chandrasekaran*[†], *P. Brighten Godfrey*[•⋆], *Haitham Hassanieh*[◇], *Bruce Maggs*[‡]
[•]*UIUC,* [†]*VU Amsterdam,* [⋆]*VMware,* [◇]*EPFL,* [‡]*Duke University and Emerald Innovations*

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: 5G's general-purpose eMBB channel has much higher bandwidth than 4G but empirically offers little improvement for common latency-sensitive applications, while its ultra-low-latency URLLC channel is targeted at only specific applications with very low bandwidth requirements.

We explore a different direction for wireless channel design to address the fundamental bandwidth-latency tradeoff: utilizing two channels – one high bandwidth, one low latency – simultaneously to improve performance of common Internet applications. We design DChannel, a fine-grained packet-steering scheme that takes advantage of these parallel channels to transparently improve application performance. With 5G channels, our trace-driven and live network experiments show that even though URLLC offers just 1% of the bandwidth of eMBB, using both channels can improve web page load time and responsiveness of common mobile apps by 16-40% compared to using exclusively eMBB. This approach may provide service providers important incentives to make low latency channels available for widespread use.

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 [21]. Emerging VR, AR, and cloud gaming applications also rely on low latency to deliver a seamless user experience. For instance, VR requires 20 ms or lower latency to avoid any simulator sickness [19].

Current mobile broadband, serving general Internet applications such as web browsing and video streaming, have not yet delivered consistent low latency performance, in part due to the inherent trade-off between latency and bandwidth [22]. 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. 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 to no improvement for latency-sensitive applications, such as web browsing [34, 35, 50]. Experimentally, web page load time in existing 5G deployments, even in close-to-ideal circumstances (a stationary device and a channel with little utilization), is similar to 4G for pages smaller than 3 MB in size and about 19% faster than 4G for pages larger than 3 MB [34]. This is due to 5G eMBB having 28 ms or larger latency, broadly similar to 4G [34]. Our measurements of 5G mmWave showed similar results, at around 22 ms in ideal conditions.

Meanwhile, 5G URLLC promises an exciting capability of very low latency, in the range of 2 to 10 ms [6], but compromises severely on bandwidth, making it unsuitable for common mobile applications. Our experiments emulating web browsing (the most widely used mobile application [44], and far from the most bandwidth-intensive application) over URLLC with 2 Mbps bandwidth show web page load times would be 5.87× worse than with eMBB. Hence, neither using URLLC alone nor using eMBB alone provides good performance. As the latency-bandwidth trade-off is fundamental, this separation between a high bandwidth channel (HBC) and a low latency channel (LLC) is likely to persist; 6G, for example, is also expected to include both [54].

We believe, however, that the availability of two channels offers an opportunity to deal with the fundamental latency-bandwidth tradeoff in a new way, beyond simple static assignment of an application to a single channel. Specifically, we argue 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 and web-based mobile applications.

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 transport-layer mechanism to combine multiple channels, MPTCP [49], 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 [42] and TAPS [38] – 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, we design DChannel, a system that leverages parallel channels to improve the performance of mobile applications. DChannel comprises two modules

running at either end of the channels – namely, in the mobile device OS and in a gateway device operated by the service provider. Central to the approach is a packet steering scheme that operates at the network layer (i.e., IP packets) without requiring any application input. Such fine-grained, per-packet decisions (as opposed to, for example, HTTP object-level steering) are key to making effective use of the limited LLC bandwidth. To decide which packets are worth accelerating, since LLC bandwidth is extremely limited, DChannel treats the channel as an expensive resource and calculates the benefit and cost of utilizing the LLC for each packet. Finally, since the parallel channels could occasionally confuse the transport layer with out-of-order delivery, DChannel employs a reordering buffer in the mobile device and gateway.

To evaluate our design with a concrete scenario, we leverage 5G's eMBB and URLLC as our HBC and LLC. We evaluate the benefit of DChannel in our experimental testbed (§4). Our testbed includes a prototype that can capture and steer application traffic, and a high-fidelity trace-driven network emulator that emulates cellular network latency variability and delay caused by radio resource control (RRC) state transitions [41]. We gather two types of real 5G eMBB traces – mmWave and lowband – in three different scenarios: stationary, low mobility, and high mobility. Our evaluations cover popular web applications such as web browsing and Android mobile applications. Using the testbed, we evaluate our packet steering scheme and compare it with prior approaches such as MPTCP [2] and ASAP [29]. We also evaluate DChannel in live 5G eMBB networks. Our key findings are as follows:

- DChannel, which requires little per-connection state and no application knowledge, yields superior performance compared to the other evaluated schemes—object-level steering, static packet-size-based steering, as well as prior work, MPTCP and ASAP [29], which used multiple channels in other settings.
- Compared with exclusively utilizing the eMBB, allocating a modest bandwidth of 2 Mbps to URLLC allows DChannel to improve web page load time (PLT). Under conditions that are ideal for eMBB (a stationary client with a line of sight to the base station and full signal strength), DChannel reduces PLT by 20% and 33% in 5G mmWave and low-band settings, respectively. Under more challenging mobile conditions, DChannel improves PLT by 37% and 42% in 5G mmWave and low-band, respectively.
- In addition to web browsing, we evaluated three Android mobile apps in a live environment and find DChannel improves apps responsiveness by 16% on average.
- Somewhat surprisingly, DChannel improves *sustained throughput* in our mobile 5G setting by roughly 10% – a useful side benefit of accelerating the TCP control loop in dynamic environments.

Finally, we discuss deployment strategies, challenges, and future opportunities. We believe our basic techniques can apply to a variety of latency-sensitive applications, and open new opportunities for app developers and cellular providers.

2 Background and Motivation

2.1 Channels in 5G

5G wireless networks are designed to support applications with very different service level requirements. The 5G standard known as New Radio (NR) specifies three service models: (1) enhanced mobile broadband (eMBB) for standard high-data-rate Internet and mobile connectivity, (2) ultra-reliable low-latency communication (URLLC) for mission-critical and latency-sensitive applications, and (3) massive machine-type communications (mMTC) for large-scale IoT deployments. We describe eMBB and URLLC in more depth.

(1) Enhanced Mobile Broadband: This service focuses on providing high-data-rate mobile access. It is considered an upgrade to 4G mobile broadband that will satisfy the ever-increasing demand for mobile and wireless data. 5G eMBB can operate either at the low-frequency bands below 6 GHz which we refer to as low-band or the high-frequency bands around 28 GHz/39 GHz which we refer to as millimeter wave (mmWave). The mmWave bands are a key new technology in 5G as they offer 10× the bandwidth that is currently available to 4G LTE networks [4], enabling user throughput of around 1 Gbps [15].

Providers like Verizon, AT&T, and T-Mobile have already deployed both the low-band and mmWave 5G in several major US cities, including Chicago, Atlanta, New York, and Los Angeles [9–11, 34]. 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 [34]. The measurements were performed, however, in conditions favorable to mmWave such as 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. This is because radio access networks (RANs) operating in the mmWave bands use very directional beams to compensate for high signal attenuation, making them vulnerable to blockage and mobility. High data rate communication is possible only when the RAN access point aligns its beam towards the user [27]. This process, commonly referred to as beam alignment, can introduce significant delays, especially when users are moving, which requires the access point to keep realigning the beam of each user [23, 27]. Furthermore, the user or other obstacles can easily block the beam, leading to unreliable and inconsistent performance both in terms of changes in throughput and highly variable RTT [3, 32, 34]. Our own experiments in Chicago also confirm this and show that the RTT can vary sig-

nificantly even for stationary clients and is further exacerbated while walking or driving. This is because 5G eMBB mainly optimizes for high data rates, focusing less on reliability and low latency.

(2) Ultra-Reliable Low-Latency Communication: Unlike eMBB, this channel focuses on providing highly reliable, very low latency communication at the cost of limited throughput. It aims to support mission-critical and emerging applications with stringent latency and reliability requirements such as self-driving cars, factory automation, and remote surgery. While the URLLC channel is yet to be deployed in practice, the standard specifies a target 0.5 ms air latency between the client and the RAN (1 ms RTT) with 99.999% reliability for small packets (e.g. 32 to 250 bytes) [15]. It also specifies a target end-to-end latency (from a client to a destination typically right outside the cellular network core) of 2 to 10 ms with throughput ranging between 0.4 to 16 Mbps depending on the underlying application [6]. URLLC is expected to operate in the sub-6 GHz frequency bands (e.g. 700 MHz or 4 GHz) and operators are expected to use network slicing to provide dedicated resources to URLLC clients in order to guarantee consistent performance in terms of latency and reliability across both the radio access network (RAN) and the cellular core [6]. Finally, client access to the URLLC channel will be controlled by the network operators. The access control network slicing mechanisms, however, are left to the operators' own implementations [8].

2.2 Web browsing traffic

While we evaluate several applications, web browsing is the major focus of this work and serves as a running example.

A single web page may contain tens to hundreds of relatively small-sized web objects distributed across multiple servers and domains. Consequently, web browsing traffic is characterized by its often short and bursty flows. 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 [48]. Fetching these web objects translates to many HTTP request-and-response interactions across many short flows. The browser fires a page load event when it finishes rendering a page, which is used to determine Page Load Time (PLT), a performance metric for web browsing. Although PLT has some shortcomings, the alternatives are not free from issues, and PLT is most widely used. PLT is typically dominated by DNS lookup, connection establishment, and TCP convergence time—which require little throughput but are highly dependent on RTT. Prior work also showed that increasing TCP throughput beyond $\approx$ 16 Mbps offers little improvement in PLT [45].

Of course, web page loading is affected by client CPU and server delay, in addition to network delay. Prior work found that 35% of the PLT is spent in client-side computations [47]. But the above characteristics, combined with the fact that mobile CPUs have been getting increasingly powerful [26], still suggest that network latency plays an important part in mobile web performance. Moreover, a significant portion of network latency lies in the "last mile" connection of the cellular network. Many other mobile apps also rely on HTTP-based interaction with cloud services, resulting in similar network performance requirements.

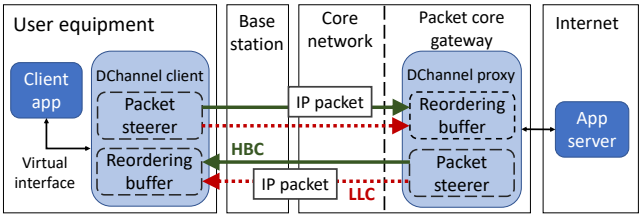

Figure 1: *The overview of DChannel. It has two main components: packet steerer that steers application traffic to LLC and HBC, and reordering buffer that reorders packets coming from LLC.*

3 DChannel Design

3.1 High-Level Architecture

To steer application traffic in both uplink and downlink channels, there will be two main components, one in the mobile client device and one in the mobile core network (Figure 1).

On the client, applications interact with the network through a network interface as usual. In our prototype, this is a special virtual TUN interface designated for traffic that should utilize both the HBC and LLC. The client-side agent captures outgoing packets on this interface and implements an algorithm to steer traffic between the two channels. The agent also captures incoming traffic on both channels and merges it into the virtual interface, after buffering it as needed to reorder packets (§3.6).

The proxy-side agent performs symmetric functions using the same algorithms – steering traffic headed towards the client, and merging and reordering traffic outbound to the Internet. This agent runs in the service provider's network, on a gateway at the point where the separate HBC and LLC channels begin. The exact location of the proxy-side agent may depend on the service provider's internal architectural choices; note that it is not necessarily located at the RAN base station, because the LLC's latency optimizations may extend into the packet core (e.g., for prioritized queuing and routing) [5].

The next subsections detail how we design the steering component, in several steps, as it is the more complex component. After that, we describe the reordering buffer.

3.2 Steering Granularity

To build the packet steering module, we begin with the question of the granularity, and corresponding layer, at which steering should occur. We considered splitting at two different layers: the application layer and the network layer.

Application-layer splitting refers to steering application requests and responses to the appropriate channels. In the context of web browsing, this approach translates to requesting and delivering web objects (in the form of HTTP requests) on either LLC or HBC. Application-layer splitting is broadly similar to Socket Intents [42].

Object-level splitting may benefit from application-level knowledge about web objects, which vary in size and priority. Since LLC is bandwidth constrained, LLC can only deliver small objects faster than HBC.[1] Web pages have complex dependency structures, and certain objects can be on the critical path for web page loading. These critical-path objects need not necessarily be small in size. Small objects might have low priorities such that accelerating them will not improve load time and thus would waste LLC bandwidth. In contrast, high-priority objects can be large such that sending those to LLC will be slower than HBC. Application-level input could help distinguish between these cases.

But object-level splitting has two drawbacks. First, we want to avoid requiring application input, which creates deployment hurdles and extra work for developers. Second, it misses opportunities for latency improvement. A web object that's not small enough to be sent over LLC will still involve small and latency-sensitive DNS lookups, TCP connection establishment, TLS handshaking, and ACKs. Accelerating this traffic could significantly reduce object delivery time. We later demonstrate (§5.3) that object-level splitting is less effective than finer-grained packet-level steering.

Steering packets at the network layer (e.g., IP datagrams) comes with its own challenges, however. First, we do not have any application-level insight into the flow: we do not necessarily know how packet-level acceleration affects application-level acceleration, so we will need a careful steering heuristic. Second, even if we identify the packets to accelerate, sending packets within a flow across two different channels might result in the packets arriving out-of-order, confusing TCP. To address this issue, we will introduce a small *reordering buffer (ROB)* at the endpoints. The following subsections discuss these components of the design.

3.3 Packet Steering Intuition

Define a "message" as a sequence of one or more packets such that the receiving endpoint can take some useful action after receiving the full message. For example, an individual SYN or ACK is a message (because the transport layer can act on it), and an HTTP request or a full response spread across multiple packets is a message (because the application may be able to process the request, display an object to the user, etc.). In contrast, an individual data packet belonging to a large HTTP request/response is not a message on its own and would not be worth accelerating individually since we need to accelerate the whole sequence of packets to finish the message.

Ideally, we would like to accelerate the delivery of messages, especially those that are most valuable to accelerate, within the bandwidth constraints of the LLC. This suggests a *cost-rewards calculation* weighing the benefit of accelerating a message against the cost of utilizing the meager bandwidth of the LLC which might be better spent on other messages.

A direct, exact cost-rewards calculation is infeasible since DChannel running at the network layer lacks full knowledge of message boundaries (in the application's data stream), as well as the relative value of messages to the receiver's transport layer or application. This leads us to begin with a permissive assumption: any packet *might* be a message boundary and we will optimistically consider accelerating it. Nevertheless, even operating transparently at the network layer, DChannel does have certain information about rewards and costs that will help it distinguish among packets.

First, the benefit of steering a packet to the LLC depends on how much its arrival time would improve, if at all, compared to using the HBC. This depends on packet size, current output queue lengths for both channels (which are locally observable), and latency of both channels (which can be estimated). In addition, the vast majority of applications utilize TCP or other transport that delivers messages in order.[2] This means that for a message inside packet P_i, *delivery of the message to the application* (as opposed to the delivery of P_i to the receiving host) will depend not only on the arrival time of P_i, but also on the arrival time of packets $P_0, \ldots, P_{i-1}$ (which can also be estimated). For example, suppose P_{i-1} was sent over the HBC, and P_i is ready to send immediately after. If P_i is also sent over HBC, the pair will arrive at about the same time. If P_i is sent over LLC, it will very likely arrive much sooner, but will end up waiting for P_{i-1} before it can be delivered to the application, meaning sending over the LLC is likely not useful in this case.

Second, the cost of utilizing LLC resources will depend on the packet length and how much the LLC will be in demand for other messages in the near future. The latter is not perfectly known, but current or recent outgoing LLC queue depths provide some signal.

The net effect of the above considerations is that packets should tend to get steered to the LLC when they are smaller, and when they are more isolated in time as individual packets or members of short packet sequences. This corresponds well with the intuition of prioritizing acceleration of control messages or small application-level messages. We now proceed to describe how we realize this cost-rewards approach.

3.4 Rewards and Cost

Problem statement. The packet steering algorithm is presented with a sequence of packets and needs to decide if each

[1] If URLLC is assigned a capacity of 2 Mbps (≈250 bytes per ms) and its RTT is ≈15 ms less than that of eMBB, any object of size larger than 3.75 KB are likely to be delivered faster on eMBB.

[2] Some don't, of course, but our goal in this work is to develop generic packet steering, leaving application-specialized schemes for the future.

packet P_n should be sent via LLC or HBC. We let $P_1, \ldots, P_n$ denote the sequence of packets in a single end-to-end flow (by which we mean a unidirectional transport layer connection, which may contain multiple messages).

Rewards. At the packet level, the objective is to minimize the *packet completion time* C_n, defined as the time by which all packets $P_0, \ldots, P_n$ would arrive at the receiver. This captures the intuition (§3.3) that any P_n might be a useful message to accelerate on its own, but it wouldn't be delivered to the application until prior packets are also delivered. The benefit of sending a packet P_n via LLC is thus the reduction of C_n if P_n is sent via LLC (denoted $C_{n,LLC}$), compared to when it is sent via HBC (denoted $C_{n,HBC}$). Thus, we calculate the rewards for sending P_n via LLC as: $R(P_n) = C_{n,LLC} - C_{n,HBC}$.

To calculate the above, we first need to estimate the delivery time D for a packet that depends on the channel/link[3] propagation delay $Dprop_{link}$ and bandwidth B_{link}, packet size, and the link's queue size Q_{link} at time t_n . The Q_{link} counts the number of bytes that have been enqueued for transmission through a *link* but have not yet been transmitted out the interface. Delivery time for P_n on a certain *link* is thus:

$$D_{link}(P_n) = Dprop_{link} + (size(P_n) + Q_{link}(t_n))/B_{link} \quad (1)$$

The packet completion time for P_n (C_n) should also account for completion times of P_0 through P_{n-1} (i.e., C_{n-1}) since P_n may arrive at the receiver before P_{n-1}, especially if P_n is sent over LLC and P_{n-1} was sent over HBC. Thus, we can calculate ($C_{n,link}$) as:

$$C_{n,link} = max(C_{n-1}, (t_n + D_{link}(P_n))) \quad (2)$$

Note that $Dprop_{link}$ are nondeterministic, comprising dynamic channel delay and any congestion along the channel's path, and will thus have to be estimated. We return to this later.

Cost. The cost of sending a packet to the LLC comes from the increased utilization of LLC. Intuitively, the cost should increase with the added queueing delay that a packet arriving very soon after P_n would experience, i.e., $size(P_n)/B_{llc}$. The cost should also be higher if the LLC is currently more highly utilized so that its limited capacity is reserved for higher-reward packets. We use a heuristic that captures this by adding these two effects; specifically, we compute the cost (or fare F) of putting P_n on LLC as:

$$F(P_n) = (size(P_n) + Q_{llc}(t_n))/B_{llc} \quad (3)$$

Note that to be more precise, we should compute the difference in costs of putting the packet on LLC vs. HBC. But as the HBC bandwidth is dramatically higher, its cost is negligible and we omit it for simplicity.

[3]We use these terms interchangeably for convenience. Note, however, the LLC channel may involve acceleration in the WAN in addition to the RAN, so it actually may span multiple physical links.

Comparing rewards and cost. At a high level, we want to steer packets to LLC when the rewards outweigh the costs, but comparing them involves a tradeoff: the benefit is immediate to packet P_n, whereas the cost affects possible subsequent packets which may not appear. We introduce a parameter α to capture this, so that we will send a packet to LLC when: $R(P_n) > \alpha F(P_n)$.

Calibrating α. If we set α too low, a flow may aggressively send packets to LLC so that it will deny resources to another flow in a multi-flow application. If we set it too high, we can be too conservative in utilizing the fast LLC. To find a good α and determine how sensitive performance is to its value, we conduct experiments with web browsing across different alpha values. We load 40 web pages from our corpus over different α values and pick α with the best Page Load Time (PLT) result on average. We use our testbed (§5.1) and apply the packet steering over HBC and LLC. For LLC, we use 5G NR URLLC as a reference where the RTT and bandwidth is 5 ms and 2 Mbps. For HBC, we vary its RTT while fixing bandwidth at 200 Mbps.

The detailed results are in §A.2. In summary, the results confirm that setting α too low or high has suboptimal performance. The best value for HBC RTT of 20 ms to 60 ms is 0.75. This RTT range covers most cases of 5G eMBB. As the RTT increases to 80 ms and higher, $\alpha = 1$ is slightly better. The difference, however, is less than 1%. We use $\alpha = 0.75$ for all subsequent experiments.

Note on design. The steering approach described here is not an optimal choice derived from a model – it is a heuristic, particularly the calculation of cost and calibration of α, in part since some of the relevant information (like the application-level importance of a particular packet) is unavailable. However: (1) we find the heuristic does perform well in realistic environments, (2) even if poor decisions do occur, they lead only to suboptimal performance, rather than a correctness problem, and (3) performance is not very sensitive to the exact value of α. In particular, even with $\alpha = 0$ – which corresponds to the greedy strategy, where each packet uses LLC whenever it expects a reward for itself – there is still a very good PLT improvement, within 5% or less of the best α. That said, this problem could be interesting to formalize in the future, perhaps as an online algorithm that could provide worst-case guarantees, or using queueing-theoretic tools.

3.5 The Packet Steering Algorithm

Putting together the above pieces, the complete steering algorithm is shown in Algorithm 1 in Appendix A.1. To make a decision, the algorithm requires (1) packet size, (2) current LLC queue size, (3) LLC bandwidth, and (4) latency of both LLC and HBC. The LLC bandwidth is controlled (assigned by the operator) so it is known, and (1) is directly observable.

LLC queue size (2) may directly be observable at the client, assuming its NIC is limited to the LLC bandwidth. But the proxy may have a higher local NIC rate. The proxy, therefore,

tracks outgoing traffic per user and computes what the queue depth would be if the NIC had been limited. Depending on the service provider's admission control policy, the rate could alternately be explicitly limited at the proxy. Client can also apply similar approach if (2) is not directly observable.

Latency (4) has to be estimated. To do this, we perform periodic handshakes (e.g., in every 500 ms in our use case). The handshakes consist of four steps, all with UDP packets: (1) the client agent sends a special packet we call a "D-SYN" to the proxy agent using both HBC and LLC. (2) The proxy agent upon seeing a D-SYN responds with "D-SYN/ACK" packets sent across both HBC and LLC. (3) The client agent receives the D-SYN/ACK packets, updates the base RTT value for both channels based on the difference between D-SYN/ACK receive time and D-SYN release time, and replies with "D-ACK" packets sent across both channels. (4) The proxy agent receives the D-ACK packets and updates the base RTT value for both channels. We use the minimum RTT value for the measurement. As we will see in the evaluation (§5), very rough latency estimates are sufficient.

The algorithm requires maintaining per-flow state, specifically to store C_{n-1}, the estimated completion time of the most recent previous packet. The proxy also stores per-user state for its queue depth calculation.

3.6 Reordering buffers at the endpoints

Splitting packets across asymmetric paths (particularly with a latency differential, as there is for LLC vs HBC) can cause out-of-order packet delivery, which can be harmful to application performance. In particular, TCP uses out-of-order packets as a signal of congestion, potentially causing retransmissions and a drop in sending rate. To solve this problem, we adopt a reordering buffer (ROB) in the receiving direction of each of our agents, to buffer packets arriving only from LLC. Note that we only buffer packets arrived from LLC as we only want to handle packet reordering caused by sending packets through the faster LLC and not to solve reordering caused by external factors such as wireless losses.

To avoid unbounded buffering delay if the previous packet was lost, the ROB also releases packets after a timeout. Ideally, the timeout should equal the latency of HBC, but because the latency of HBC can be variable and hard to track, we use a conservative 100 ms timeout. We evaluate the effectiveness of this timeout value under random packet loss in §5.

4 Prototype and Experimental Setup

Our experiments involve a client representing a mobile end-user application (e.g., a web browser) fetching content from a web or content server. Both the client and server endpoints have access to two interfaces, one representing the high-bandwidth channel (HBC) and the other the low-latency channel (LLC). In the case of 5G, HBC and LLC map to eMBB and URLLC, respectively. Depending on the experiment conditions, the interfaces may be real or emulated. We masked the two interfaces at the endpoints, however, using a smart DChannel virtual interface implemented on top of a `TUN` device; the client and server use only this virtual interface to send and receive data. Our DChannel prototype then performs endpoint-transparent (and application-agnostic) steering of traffic.

We developed a DChannel prototype and packaged it as a UNIX shell, similar to the shells in Mahimahi [36]. The shell captures all outgoing traffic from any *unmodified* application running within it and tunnels them to our DChannel implementation; it processes incoming traffic in a similar application-transparent manner, so both the steering and buffering modules of DChannel are used. Our DChannel prototype attaches additional metadata (sequence number and flow ID) prior to transmission to assist the receiver in reordering packets and strips this before delivering to the application. We used our own metadata header as a convenience, but in a real implementation, this could be avoided by looking inside the layer 4 header.

We evaluated the performance of DChannel using this prototype under two settings. The first is a *live* setting where we used the actual 5G NR eMBB channel as HBC. The second setting, in contrast, is one where we emulated the eMBB channel based on traces that we gathered from an actual 5G eMBB channel. In both settings, since URLLC is not yet commercially available, we emulated its "expected" behavior (based on the 5G specification [6]) using a low-latency, bandwidth-limited wired Ethernet connection.

4.1 Live-eMBB Setting

In this setting, DChannel steers traffic over two real interfaces (Fig. 2): One interface is tethered with a 5G phone for providing access to a *live* eMBB channel, while another is connected to a low-latency bandwidth-limited Ethernet connection for emulating the URLLC channel. Packets transmitted over the 5G eMBB channel traverse the core network of the mobile provider before exiting via the packet gateway (i.e., *mobile path*) and then one or more ASes in the public Internet (i.e., *Internet path*) to reach our server. Data sent over the Ethernet interface, in contrast, traverse a traditional ISP and then one or more ASes to reach the server. On the server side, DChannel receives all the packets from both the interfaces, reorders them (if required), and then delivers them to the server-side application via the `TUN` device.

We used Ethernet and not WiFi for emulating URLLC, since the channel is expected to provide high reliability (≥ 0.9999) [8]. We capped the bandwidth of this link using `netem` to emulate the low bandwidth of URLLC. Since the client must remain physically plugged in to a wired network for emulating URLLC, this setting allows us to study performance only in stationary conditions.

4.2 Emulated-eMBB Setting

To evaluate DChannel under a wide variety of scenarios, specifically those including client mobility, we used *trace-*

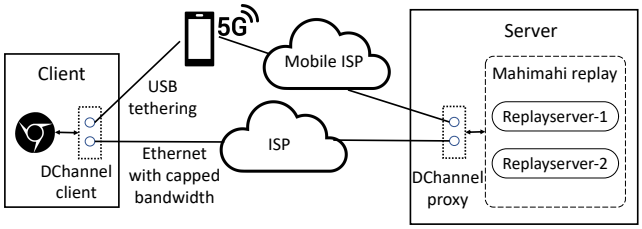

Figure 2: *In our live 5G eMBB testbed, the client has two paths to the server: One path over a tethered connection to a 5G phone for utilizing the eMBB channel, and the other through a bandwidth-capped connection over Ethernet, for emulating the URLLC channel.*

driven emulations. Below, we describe how we captured the network (latency and bandwidth) traces of the 5G eMBB channel under stationary and low-to-moderate mobility scenarios and used them in our emulations.

4.2.1 Collecting network traces

To capture the temporal variability of mobile networks, we measured both the latency and throughput of the eMBB channel over time.

Latency traces. We measured the latency of the eMBB channel by periodically sending probes (UDP packets) from the client to the server. We set the probing period to 15 ms to force the UE radio to remain always in "active" mode and generate only a small amount of probe traffic to avoid queuing. Our measurements capture the latency imposed by the base station and core network, since our server was always in close proximity to the client (i.e., less than 150 miles), minimizing the Internet-path latency. Our `traceroute`s from the client to the server, although not shown in the paper, also confirmed that the latency between the client and the server was very close to the latency between the client and the packet gateway.

Bandwidth traces. We measured the throughput across time of both uplink and downlink channels by saturating them with MTU-sized UDP packets. Since TCP cannot reliably saturate the highly variable cellular uplink and downlink concurrently, we used an overestimated fixed sending rate to always fill the queue. First, we measured the maximum supported upload and download UDP throughput using existing tools such as `iperf`. Then, we sent traffic at this maximum rate from both endpoints. Finally, we used the actual packets received over time by the endpoints to estimate the uplink and downlink capacities.

Measuring both latency and bandwidth. A key challenge in measuring both latency and bandwidth simultaneously is avoiding interference: bandwidth-intensive operations can saturate the link and fill the queue, thereby inflating the latency. Since cellular networks use per-user queues, we addressed this challenge by measuring latency and bandwidth from separate devices. When using two separate devices, we did not see any perceivable interference for measurements on 5G low-band, although we observed them on 5G mmWave. Specifically, we observed inflation in latency if a nearby device was uploading data at more than 5 Mbps using mmWave.[4] For 5G mmWave, we measured, hence, only the downlink throughput over time; we set the uplink bandwidth to a single, fixed rate of 60 Mbps.

The accuracy of temporal variations in latency matters *most* for our trace-driven emulations, since the main applications that we use in our evaluations, web browsing and mobile apps, are latency-sensitive. The performance of such applications crucially depends on TCP-related configurations (e.g., initial congestion window) and network latency (or RTT) rather than on available bandwidth, particularly when the bandwidth is more than 16 Mbps [45]. Our approach to estimating bandwidths, therefore, is adequate for our evaluations.

4.2.2 Emulating the traces

In the emulated-eMBB setting, we run both the client and the server on the same machine. DChannel then steers traffic between them over two virtual interfaces, emulated using an extended version of Mahimahi [36]. Specifically, we extended Mahimahi's delay shell to vary the eMBB channel latency over time, based on a trace generated from a real 5G deployment. The modified delay shell accepts a trace comprising a "timeline" of RTT values and halves each value to derive the individual uplink and downlink latency timelines. The shell then assigns per-packet latency by choosing an uplink or downlink latency by matching the time a packet arrives at the interface against the timelines. Since the trace-file granularity is one RTT sample per 15 ms, we use linear interpolation for assigning RTTs arriving between two samples. Similarly, we emulated URLLC with a propagation delay of 5 ms and bandwidth of 2 Mbps, unless noted otherwise.

Mobile applications' traffic (especially web browsing) is typically bursty in nature and contains periods of inactivity. To preserve energy during *idle* periods, UEs switch to a low-power (or "sleep") state, which supports discontinuous reception (DRX). The transition to the low-power state depends on an *inactivity timer* that we observed (through probing [35]) to be around 30 ms for 5G mmWave; once the device enters this state, it will "wake up" periodically (every 40 ms). When emulating the latency traces, we therefore also estimate the radio power states of the device (based on its activity) and take into account any additional latency the state transitions may impose. A packet that arrives 20 ms after the UE enters the sleep state, for instance, will experience an additional 20 ms delay before it is processed. This delay, however, is not incurred on the uplink. For 5G low-band, we set the inactivity timer to 100 ms and wake-up interval to 20 ms.

For the bandwidth emulation, we extended Mahimahi's link shell to emulate a time-varying bandwidth that changes every second. To emulate a link of capacity 60 Mbps at time

[4] Low-band uses OFDMA so multiple devices can communicate at the same time and the latency is not inflated, while mmWave uses single carrier modulation, where multiple devices must take turns transmitting and the antenna must switch its beam pattern.

Table 1: *Characteristics of network traces gathered from actual 5G deployments at different locations and under different conditions. 'p50' and 'p98' refer to the 50th and 98th percentiles, and 'CV' refers to the coefficient of variation.*

Trace name	***Span (mins.)***	***RTT (ms)***					***Mean bw. ↑/↓ (Mbps)***	***Description***
		min.	***p50***	***p98***	***mean***	***CV***		
mmWave-Stationary (MM-S)	60	18	22	106	29.88	0.77	60/140	UE was in a building in the downtown Chicago, placed near a window with a base station in line of sight.
mmWave-Walking (MM-W)	56	16	22	120	30.32	0.98	60/110	UE was held by user walking in downtown area of Chicago.
mmWave-Driving (MM-D)	18	18	40	236	56.15	0.96	60/100	Phone was with a user driving through the downtown area of Chicago at low to moderate driving speeds.
LowBand-Stationary (LB-S)	60	34	40	132	45.20	0.50	26/93	Phone was located in a building in a university campus. It was placed near a window with full signal strength.
LowBand-Walking (LB-W)	53	32	52	156	58.94	0.50	21/63	Phone held by user walking in a university campus.
LowBand-Driving (LB-D)	23	34	54	202	68.84	0.62	15/57	Phone was with a user driving near a university campus.

n seconds, for instance, this extended link shell will release 7.5 KB per millisecond. In our emulation tests, we also used a FIFO (drop-tail) queue, and we set the buffer to 800 MTU-sized packets.

5 Evaluation

We evaluated DChannel using 5G eMBB and URLLC as HBC and LLC, respectively. We ran the client (e.g., a web browser) on a laptop, unless otherwise mentioned. The laptop had 16 GB RAM, 512 GB SSD, and an Intel Core i7 processor running Ubuntu 20.04 (Focal Fossa).

5.1 Testbed Configuration

In the live-eMBB setting, we tethered the laptop with a Google Pixel 5 phone using USB (refer Fig. 2). We ran the live experiments from two locations: UIUC campus with access to 5G low-band and the Chicago downtown area for 5G mmWave access. We emulated the URLLC link between the client and server using a wired (Ethernet) link and configured it based on URLLC end-to-end specification and use-cases [6]. The emulated link provides 5 ms RTT between the client and the network gateway and has 2 Mbps capacity. At the 5G mmWave test site, however, the wired link only provided a minimum latency of 8 ms for the URLLC emulation.

We also collected latency and bandwidth traces (summarized in Tab. 1) at the two test locations under three mobility conditions: stationary, walking, and driving. All traces were collected using Google Pixel 5 phones with Verizon 5G. Though mmWave offers lower latency (for eMBB) than low-band, it also experiences higher variance than low-band, even when the UE was stationary. This inconsistency in performance becomes even worse under mobility. Low-band offers a stable, albeit relatively higher, RTT than mmWave. We ran all the components, i.e., the client, DChannel's modules, and the server, on the laptop in the emulated-eMBB setting, and used the traces (in Table 1) for emulating the eMBB channel.

5.2 Application use cases

We evaluated DChannel on 5G under a wide variety of network conditions to highlight its benefits for web browsing and web-based mobile (Android) applications. We supplemented these experiments with a bulk-download application for demonstrating DChannel's merits for long (i.e., bandwidth-intensive) flows.

Web browsing. To measure the improvements brought about by DChannel for web browsing, we first fetched a set of 200 web pages of "popular" websites, selected uniformly at random from the Hispar corpus [16]. The sample comprised 40% of landing and 60% of internal pages from 200 websites. The median web page size and the number of objects are 3.7 MB and 60, while the 95th percentile are 11.8 MB and 168. When fetching these pages, we recorded all the HTTP requests and responses using mitmproxy [1]. Then we used a version of Mahimahi with HTTP/2 support [53] to serve the responses from our server. While recording the pages, we also estimated the server response time for each request by subtracting the *time-to-first-byte (TTFB)* from the client-to-server RTT. We used the server-response times to emulate server-side processing delays during the replays.

We used an unmodified Chromium browser spawned within a DChannel shell to fetch the pages from our server. We cleared the browser and DNS caches prior to each fetch and used the default Linux congestion control, TCP CUBIC, unless noted otherwise, for all web-browsing experiments. We measured the *page-load time (PLT)*, based on the `onLoad` event [37] in each experiment, on each fetch. In the live-eMBB setting, we first used the DCHANNEL scheme to fetch a page and repeated that page fetch in quick succession using a different scheme. We calculated the difference in PLT between the different schemes and repeat the fetch five times to compute the mean difference in PLTs. In the emulated-eMBB setting, we picked a random sub-sequence from a trace for each page fetch. Given a page, we used the same sub-sequence for measuring the PLT across different schemes

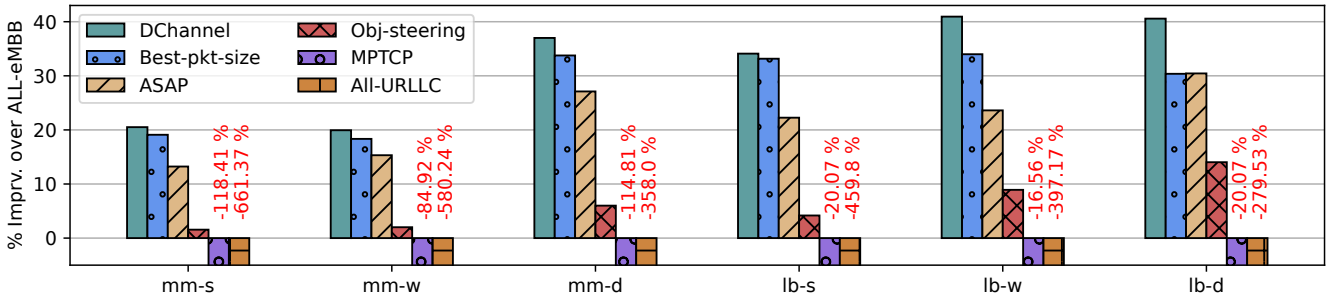

Figure 3: *DChannel offers at least 20% lower PLT compared to that of All-eMBB, and it performs better than all other schemes. MPTCP's PLTs are 17% to 118% worse than when using a single eMBB channel across all traces.*

and computed the difference in PLTs. We used the mean of the PLT differences across five trials, with each using a random sub-sequence of the trace, for comparing and contrasting the different schemes.

Mobile application. We downloaded three popular Android applications from the Google play store, one each from three categories: social (Reddit), shopping (eBay), and news (CNN). We ran the applications on the phone (Google Pixel 5), but tunneled all network traffic to a DChannel shell running on our laptop. We then used the DChannel to steer the traffic over the appropriate channels. We evaluated the mobile applications only under the emulated-eMBB setting. Note that this setup may underestimate the performance improvements because of the additional overheads in tunneling the traffic from the phone to the laptop. We measured the RTT from our laptop to the application server to be 12 to 30 ms, which is significant given that our emulated URLLC and eMBB RTT can go as low as 5 ms and 16 ms.

To evaluate application performance, we calculated the *interaction response time (IRT)* [40]. IRT measures the time elapsed between when a user performs an interaction to when the end screen for that interaction is completely rendered. We automated user interactions with AndroidViewClient [13] and cleared application caches within each trial. We recorded the phone screen using `FFmpeg` [14] during each interaction and identified when the screen stopped changing visually using `scenecut-extractor` [12]. Since we do not control the servers used by the application, we repeated the interaction experiments with DChannel and other schemes, one after the other, in quick succession. We used the median IRT across ten trials to compare the performance of different schemes.

Bulk download. The setup for this use case is similar to that for web browsing. We simply used `curl` as the client to download a file hosted in our (Mahimahi) server and repeated the download five times.

5.3 Comparing steering schemes

We compared the PLT of DChannel's DCHANNEL scheme with that of five other schemes across multiple scenarios (using the traces in Tab. 1) in the emulated-eMBB setting.

Table 2: *Comparing the performance of DChannel with All-eMBB and All-URLLC when fetching the (*182 KB*) landing page of* `amazon.com`.

Perf. metric	*All-eMBB*	*All-URLLC*	*DChannel*
DNS lookup (ms)	44	8	8
TCP connect (ms)	42	6	6
TLS connect (ms)	53	30	30
Object transfer (ms)	209	809	144
Total load time (ms)	349	853	189

The other schemes are as follows. **All-URLLC** steers all traffic over URLLC. **Obj-steering** requests web objects (requests and responses) on URLLC whenever its fetching time is smaller than eMBB. **Best-pkt-size** steers packets whose size is lower than the *best* predefined threshold to URLLC. **MPTCP** uses the two channels concurrently, by running the Linux kernel implementation of MPTCP [2] with the default configuration. **ASAP** [29] was designed for satellite networks. It diverts traffic from satellite links (or eMBB in our case) to 3G or 4G (or URLLC) since the latter has lower latency with a higher cost per byte than the former. ASAP code is not publicly available, so we used our in-house version.

Fig. 3 compares the relative difference in PLT of each scheme compared to that obtained when only using eMBB (i.e., All-eMBB). DChannel offers the best performance under all network conditions in our emulations. In stationary scenarios (MM-S and LB-S), it improves PLT by about 20% when using 5G mmWave and 35% with 5G low-band. These improvements in 5G mmWave and low-band correspond to absolute reductions in PLTs of about 290 ms and 642 ms, respectively, compared to the All-eMBB scheme. Per Fig. 3, DChannel's performance increases in scenarios that involve UE mobility (MM-W, MM-D, LB-W, and LB-D). The PLT improvements are higher for low-band than mmWave since the former exhibits higher latency than mmWave (refer Tab. 1). We examined the relation between DChannel's performance and eMBB latency in detail in §A.3.1.

Where do DChannel's gains come from? To illustrate the an-

swer to this question, we used `curl` to fetch the landing page (i.e., the root document, "/") of `amazon.com` in the emulated-eMBB setting (40 ms RTT and 200 Mbps) without any server-side response delay. DChannel performs better than not only all-eMBB, but also all-URLLC (Tab. 2). While URLLC has lower latency than eMBB, it also has a significantly lower bandwidth than eMBB. We identified three key sources of performance gains by analyzing DChannel's per-packet decisions. DChannel steers three types of packets over URLLC: (1) DNS packets; (2) *control* packets (e.g., `SYN` and client-to-server `ACK` packets); and (3) small *data* packets. Sending DNS and SYN packets over URLLC reduces DNS lookup and TCP connection-setup times, and accelerating ACK packets reduces the object transfer time. The last category includes small data transfers such as the TLS client-key exchange and HTTP requests.

Packet vs. web objects steering. Obj-steering performs HTTP request and response on URLLC when the web object size is small such that it will finish faster than eMBB. The scheme only offers slight improvement to PLT (2-14%). This is because not all small objects are critical. In fact, we found out that only 14% of small web objects have VeryHigh priority [25].

DChannel vs. static packet-size-based steering. Best-pkt-size steers individual IP packets whose size is smaller than the *best* static threshold to URLLC, and the reordering buffer will reorder any out-of-order packets. To find the best threshold, we performed web page load experiments for each trace with five different (size) thresholds: 250, 500, 750, 1000, 1250, and 1400 bytes. We found 750 bytes and 1000 bytes give the best-averaged result (across the stationary, walking, and driving scenarios) for mmWave and Low-Band traces, respectively.

DChannel shows an overall better improvement than best-pkt-size across different network conditions, albeit best-pkt-size offers similar improvements in stationary traces. In stationary traces, network latency is more predictable, and static decisions might suffice. When the network conditions are more variable, such as in the driving scenario, however, the static decisions do not suffice. DChannel observes network conditions, as they evolve, and estimates URLLC channel usage to make steering decisions dynamically. DChannel will not steer small packets to URLLC, for instance, if the channel is already congested. Note that these results are overly generous to best-pkt-size, for comparison purposes: the best size is selected in retrospect after running on the test scenarios. In reality, determining a single packet size threshold would be complicated: it depends on application traffic patterns as well as network conditions.

Is MPTCP not designed to exploit multiple channels? MPTCP works at the transport layer, and in general, it load-balances application traffic among the available paths and aggregates their throughput. In our evaluations, MPTCP performs worse (by inflating PLTs between 16% and 66% across different conditions[5]) than simply using only the eMBB path. This poor performance stems from MPTCP's default scheduler (*minRTT*), which prefers the path with the smallest estimated RTT. This scheduler thus infers that URLLC is better than eMBB and diverts traffic to URLLC until experiencing congestion. DChannel, unlike MPTCP, works at the network layer such that it allows steering data packets on eMBB and ACK packets on URLLC. MPTCP cannot perform such packet-level steering, since it results in each path having a separate data-ACK loop, which MPTCP cannot support.

Table 3: *The p50 and (p95) of the avg. and max. buffer sizes (in bytes) when loading 200 web pages under MM-S and LB-D traces.*

	MM-S		***LB-D***	
	Proxy	***UE***	***Proxy***	***UE***
Avg buffer size (b)	2 (15,7)	12 (63.5)	14 (96)	130 (2638)
Max buffer size (b)	392 (1122)	944 (2597)	757 (2375)	2848 (15521)

DChannel vs ASAP. ASAP identifies the different phases of a web transaction (e.g., TLS handshake and HTTP request) and accelerates packets of latency-sensitive phases. It accelerates, for instance, TLS/SSL handshake as well as HTTP request traffic, but leaves HTTP responses to eMBB. ASAP performs better than all other schemes except DChannel. It falls behind DChannel, however, because of its static heuristics (e.g., accelerate all HTTP requests). HTTP requests are typically, but not always, small. A user uploading a photo, for instance, is one example where the assumption fails to hold. ASAP also encounters problems when the user browses complex internal pages that push some data to the server.

In the above experiments, we emulated URLLC based on the 5G standard. We found, however, that DChannel continues to offer significant PLT improvements even if URLLC latency is doubled or tripled or when URLLC latency changes over time (§A.3.2). DChannel offers good performance even in situations we cannot accurately estimate the eMBB RTT (§A.3.4), which is crucial for calculating rewards and cost (§3.4). We also examined DChannel's performance under different URLLC bandwidths in §A.3.3. Finally, we evaluated DChannel's rewards calculation accuracy in §A.4

5.4 Live 5G Experiments

We repeated the web-page fetches (similar to those in §5.3) over both the live-eMBB and emulated-eMBB settings. We then compared the relative improvements in PLTs brought about by DChannel across these settings, for both 5G mmWave and Low-Band (Fig. 4). In conclusion, the PLT improvements are quite similar between the live and

[5]We clipped the bottom of the Y-axis in Fig. 3 to focus on performance gains.

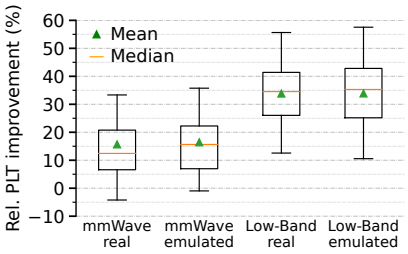

Figure 4: *Comparison of relative PLT improvements in live-eMBB and emulated-eMBB settings*[6].

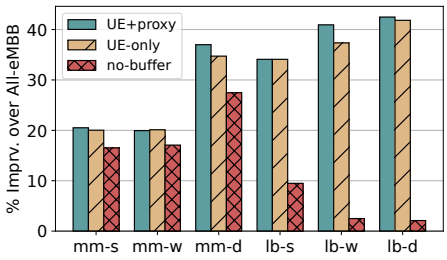

Figure 5: *Adding reorder buffer to DChannel significantly improves web page load time.*

emulated-eMBB settings, albeit the mean PLTs in the former are higher than those in the latter. The mean PLT for the All-eMBB scheme in the live-eMBB setting for 5G Low-Band is 1718 ms, while that in the emulated-eMBB setting is 1522 ms. The high mean PLTs in the emulated-eMBB settings could be due in part to the limitations of the setup. The emulations do not capture all eMBB network characteristics such as out-of-order delivery. Moreover, the web server is hosted on dedicated hardware in the live-eMBB setting, whereas in the emulated-eMBB setting it runs on the same laptop as the client. These minor setup differences, however, do not affect our claims concerning the relative performance improvements (which are similar across the two settings).

5.5 Evaluating the reordering buffer

We evaluated the effectiveness of the reordering buffer (ROB) by comparing the web browsing performance of DChannel with and without the buffer. Along with the other experiments, this evaluation uses the default TCP CUBIC, which is sensitive to in-order packet delivery. Fig. 5 shows that without the buffer, the mean PLT improvement is decreased by up to 40% (in LB-D). Overall, DChannel without the buffer will be worse when the gap between eMBB and URLLC latency increases (implied in the figure as DChannel without the buffer is performing much worse in 5G Low-Band than mmWave) because DChannel will be more aggressive in offloading packets to URLLC, causing more out of order packets.

If ROB is implemented only on the UE (downlink), PLT improvement is only reduced by 2%. The decrease in PLT improvement is because DChannel tends to offload most web browsing uplink traffic to URLLC as the client requests are generally small such that minimal out-of-order persists. Our buffer analysis in Tab. 3 confirms that DChannel proxy does not use the buffer as frequently as the DChannel client (UE). Tab. 3 also shows that ROB requires only little memory as we do not have to buffer much URLLC traffic.

DChannel under random packet drop. Since ROB holds packets from URLLC for a fixed (100 ms) timeout when they are not in order, DChannel may suffer when packet loss happens as it may delay the packet loss signal, which can happen under certain conditions (§3.6). To evaluate this effect, we

[6]The mean PLT improvements of mmWave is lower than what we reported on MM-S in Fig.3 because we used 8 ms of URLLC RTT (rather than 5 ms).

Table 4: *PLT under different random packet drop rates.*

Loss	*All-eMBB (ms)*		DCHANNEL *(ms)*	
	MM-S	*MM-D*	*MM-S*	*MM-D*
0.0%	1108	1899	883 (20%)	1096 (42%)
0.1%	1203	1963	1011 (16%)	1311 (34%)
1.0%	2643	3421	2502 (5%)	3072 (10%)

investigated DChannel performance under stationary and driving traces in the case of random packet drops. We applied a stochastic packet drop in the uplink and downlink channels with packets being dropped in both eMBB and URLLC. This reflects the case where packet drops happen on the Internet path. Note that 5G eMBB generally exhibits low packet loss rates with the 99th percentile being 1.2% [34]. Per Tab. 4, DChannel is quite resilient to random packet loss, offering better performance than All-eMBB even under high loss rates. We also investigated the interplay between DChannel and latency-based congestion control algorithms in Appendix A.3.5.

5.6 Bulk download performance

Although our primary focus is latency-sensitive applications, we also evaluated how DChannel performed for a bandwidth-intensive use–bulk HTTP transfer of a file. Fig. 6 shows the download time for various file sizes using the mmWave driving (MM-D) trace. As expected, DChannel gets the largest improvement for small objects. But interestingly, DChannel also usefully improves large object download. This is because all control packets including TCP ACKs are accelerated in URLLC, which reduces the control loop delay, helping the transport layer adapt to bandwidth changes more quickly. Specifically, we found DChannel resulted in better utilization of the available eMBB throughput by $\approx 10\%$ when there are large throughput variations (e.g., in the driving scenario).

5.7 Mobile application performance

Fig. 7 shows the application response time improvement of DChannel across three common user interactions that require communication to the server. On average, DChannel improves the response times of application launch (15%), query searching (12%), and information (e.g., product or news) loading (21%). It is unsurprising that query searching gives lower improvement since it incurs higher server-side delay. The overall

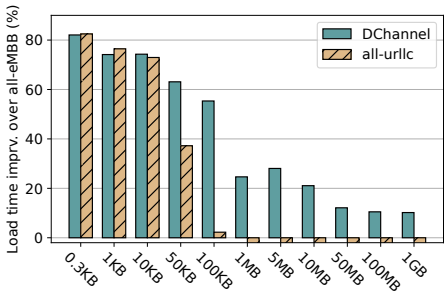

Figure 6: *Download time improvement of variable-sized data under HTTP. The experiment used the MM-D trace with the buffer set to 800 packets (≈ 2× trace BDP).*

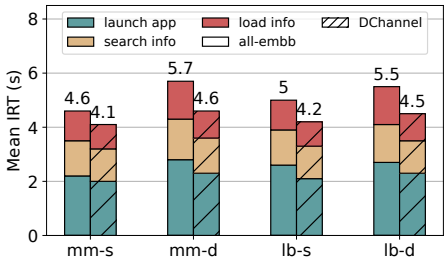

Figure 7: *Android mobile application interaction response time (IRT) of All-eMBB and DChannel when performing three different tasks. We averaged the result from three applications.*

mobile apps improvement is lower than the web browsing in part to our experimental setup (§5.2).

6 Discussions and Future Work

Deployment for 5G networks: DChannel requires cellular operator support, to allow URLLC for non-critical traffic and to perform stateful packet steering. However, operators may omit some of the DChannel implementations to make deployment easier, such as eliminating the reordering buffer (ROB) on the core gateway since DChannel shows only minor performance degradation without ROB in the proxy-side (§5.5). DChannel stateful packet steering may not be simple to implement especially when there are multiple gateways. We leave this to future work.

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 writing, URLLC is not yet deployed in public. However, based on the white paper [7], 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/km^2, connection density of 100,000/km^2, and max end-to-end latency of 10ms. Thus, the 2 Mbps maximum bandwidth per user for general application traffic used in our experiments 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, 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 small increases in URLLC latency (§A.3.2).

Resource contention among applications: Multiple applications inside a user device 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 [6]. Expanding URLLC applications 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.

Emulation uncertainty: The real URLLC performance might not match our emulated URLLC that follows the 5G NR white paper. However, we have performed several experiments to show the robustness of DChannel under variable URLLC conditions. Emulating the real behavior of a cellular network (eMBB) is also a known hard problem [51], and our approach of using two phones to capture both eMBB latency and bandwidth might not be perfect. We have compared DChannel performance with the emulated eMBB and live eMBB in stationary conditions and conclude that DChannel offers the same performance benefit (§5.4). However, we have not yet evaluated DChannel under non-stationary live eMBB due to the environment limitation (§4.1).

Other applications: 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 single channel. For instance, cloud gaming, which allows users to play games from remote servers, requires high bandwidth to stream 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 [17, 39]; 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 [24] or specialty [18] 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 Related work

There have been multiple works that try to exploit the multi-access connectivity on the client.

Application layer multipath: Socket Intents [42] and Intentional networking [28] both expose custom APIs to applications and offer OS-level support for managing multiple interfaces. Both of them regulate application traffic based on application-specific information. Our work, in contrast, does not require application inputs or modifications, although in the future we might consider giving input to the steerer to support more specific applications.

Transport layer multipath: There are already numerous efforts to design multipath transport protocols such as R-MTP [33], pTCP [30], mTCP [52], SCTP multihoming [31], and MPTCP [49]. These protocols deliver application traffic through multiple paths to achieve better throughput and reliability. Due to the bandwidth aggregation focus, multipath transport protocols give notable benefits to long-flow dominated applications but not to short-flow dominated applications such as web browsing [20]. Our approach works transparently with single-path transport protocols (e.g., TCP and UDP).

Network layer multipath: Tsao and Sivakumar [46] proposed a super aggregation concept where TCP can achieve better WiFi throughput by selectively steering packets to 3G. ASAP [29] steers network packets over satellite ISP and lower-latency terrestrial networks to improve HTTPS. We compared DChannel against ASAP in our evaluation and found that DChannel is better for eMBB and URLLC pairs as it benefits from finer-grained decisions.

An early version of DChannel was presented in [43]. This work comes with a new and better-performing packet steering algorithm, a more robust evaluation with real-world traces and live 5G eMBB, and new use cases including mobile apps and bulk transfer.

Acknowledgements

We thanked the anonymous reviewers and our shepherd Fadel Adib for their valuable inputs. This work was supported by a gift from T-Mobile and NSF CNS Awards 1763742 and 1763841.

References

[1] mitmproxy. https://mitmproxy.org/. [Last accessed on April 18, 2022].

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

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

[4] 3GPP Release 15. https://www.3gpp.org/release-15, April 2019. [Last accessed on May 24, 2020].

[5] 3GPP TR 23.725 version 16.2.0 Release 16. https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3453, June 2019. [Last accessed on January 20, 2022].

[6] 3GPP TR 38.824 Release 16. https://www.3gpp.org/release-16, March 2019. [Last accessed on June 16, 2020].

[7] 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, March 2019. [Last accessed on January 20, 2021].

[8] 3GPP Release 16 Description; Summary of Rel-16 Work Items. https://www.3gpp.org/release-16, March 2020. [Last accessed on June 16, 2020].

[9] AT&T: 5G Coverage Map. https://www.att.com/5g/coverage-map/, 2020. [Last accessed on June 13, 2020].

[10] T-Mobile: The Only Nationwide 5G Network Coverage Map. https://www.t-mobile.com/coverage/5g-coverage-map, 2020. [Last accessed on June 13, 2020].

[11] Verizon: 5G Coverage Map. https://www.verizon.com/5g/coverage-map/, 2020. [Last accessed on June 13, 2020].

[12] Scenecut extractor. https://github.com/slhck/scenecut-extractor, December 2021. [Last accessed on April 15, 2022].

[13] AndroidViewClient. https://github.com/dtmilano/AndroidViewClient, March 2022. [Last accessed on April 15, 2022].

[14] FFmpeg. https://ffmpeg.org/, January 2022. [Last accessed on April 15, 2022].

[15] 3rd Generation Partnership Project. Study on scenarios and requirements for next generation access technologies. Technical report, 2017.

[16] Waqar Aqeel, Balakrishnan Chandrasekaran, Anja Feldmann, and Bruce M Maggs. On landing and internal web pages: The strange case of jekyll and hyde in web performance measurement. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2020.

[17] Jozef Babiarz, Kwok Chan, and Fred Baker. Configuration guidelines for diffserv service classes. *Network Working Group*, 2006.

[18] Debopam Bhattacherjee, Waqar Aqeel, Sangeetha Abdu Jyothi, Ilker Nadi Bozkurt, William Sentosa, Muhammad Tirmazi, Anthony Aguirre, Balakrishnan Chandrasekaran, P. Brighten Godfrey, Gregory Laughlin, Bruce Maggs, and Ankit Singla. cISP: A Speed-of-Light Internet Service Provider. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2022.

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

[20] Shuo Deng, Ravi Netravali, Anirudh Sivaraman, and Hari Balakrishnan. WiFi, LTE, or both? Measuring multi-homed wireless internet performance. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2014.

[21] Yoav Einav. Amazon found every 100ms of latency cost them 1% in sales, January 2019.

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

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

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

[25] Sergio Gomes. Resource prioritization – getting the browser to help you. https://developers.google.com/web/fundamentals/performance/resource-prioritization, June 2020. [Last accessed on June 12, 2020].

[26] Matthew Halpern, Yuhao Zhu, and Vijay Janapa Reddi. Mobile cpu's rise to power: Quantifying the impact of generational mobile cpu design trends on performance, energy, and user satisfaction. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 64–76. IEEE, 2016.

[27] Haitham Hassanieh, Omid Abari, Michael Rodriguez, Mohammed Abdelghany, Dina Katabi, and Piotr Indyk. Fast Millimeter Wave Beam Alignment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2018.

[28] Brett D Higgins, Azarias Reda, Timur Alperovich, Jason Flinn, Thomas J Giuli, Brian Noble, and David Watson. Intentional networking: opportunistic exploitation of mobile network diversity. In *Proceedings of the 16th annual international conference on Mobile computing and networking (MobiCom)*, 2010.

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

[30] Hung-Yun Hsieh and Raghupathy Sivakumar. pTCP: An end-to-end transport layer protocol for striped connections. In *Proceedings of the 10th IEEE International Conference on Network Protocols (ICNP)*, 2002.

[31] Janardhan R Iyengar, Paul D Amer, and Randall Stewart. Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths. *IEEE/ACM Transactions on networking (ToN)*, 14(5), 2006.

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

[33] Luiz Magalhaes and Robin Kravets. Transport level mechanisms for bandwidth aggregation on mobile hosts. In *Proceedings of the 9th IEEE International Conference on Network Protocols (ICNP)*, 2001.

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

[35] Arvind Narayanan, Xumiao Zhang, Ruiyang Zhu, Ahmad Hassan, Shuowei Jin, Xiao Zhu, Xiaoxuan Zhang, Denis Rybkin, Zhengxuan Yang, Zhuoqing Morley Mao, et al. A variegated look at 5G in the wild: performance, power, and QoE implications. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2021.

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

[37] Jan Odvarko. Har 1.2 spec, 2007.

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

[39] Maxim Podlesny and Sergey Gorinsky. RD network services: differentiation through performance incentives. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2008.

[40] Murali Ramanujam, Harsha V Madhyastha, and Ravi Netravali. Marauder: synergized caching and prefetching for low-risk mobile app acceleration. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2021.

[41] Sanae Rosen, Haokun Luo, Qi Alfred Chen, Z Morley Mao, Jie Hui, Aaron Drake, and Kevin Lau. Discovering fine-grained RRC state dynamics and performance impacts in cellular networks. In *Proceedings of the 20th annual international conference on Mobile computing and networking (MobiCom)*, 2014.

[42] Philipp S Schmidt, Theresa Enghardt, Ramin Khalili, and Anja Feldmann. Socket intents: Leveraging application awareness for multi-access connectivity. In *Proceedings of the 9th ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2013.

[43] William Sentosa, Balakrishnan Chandrasekaran, P Brighten Godfrey, Haitham Hassanieh, Bruce Maggs, and Ankit Singla. Accelerating mobile applications with parallel high-bandwidth and low-latency channels. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications*, pages 1–7, 2021.

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

[45] Srikanth Sundaresan, Nick Feamster, Renata Teixeira, and Nazanin Magharei. Measuring and mitigating web performance bottlenecks in broadband access networks. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2013.

[46] Cheng-Lin Tsao and Raghupathy Sivakumar. On effectively exploiting multiple wireless interfaces in mobile hosts. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2009.

[47] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. Demystifying page load performance with wprof. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013.

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

[49] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. Design, implementation and evaluation of congestion control for multipath tcp. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2011.

[50] Dongzhu Xu, Anfu Zhou, Xinyu Zhang, Guixian Wang, Xi Liu, Congkai An, Yiming Shi, Liang Liu, and Huadong Ma. Understanding operational 5G: A first measurement study on its coverage, performance and energy consumption. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2020.

[51] Francis Y Yan, Jestin Ma, Greg D Hill, Deepti Raghavan, Riad S Wahby, Philip Levis, and Keith Winstein. Pantheon: the training ground for internet congestion-control research. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 731–743, 2018.

[52] Ming Zhang, Junwen Lai, Arvind Krishnamurthy, Larry L Peterson, and Randolph Y Wang. A Transport Layer Approach for Improving End-to-End Performance and Robustness Using Redundant Paths. In *USENIX Annual Technical Conference (ATC)*, 2004.

[53] Torsten Zimmermann, Benedikt Wolters, Oliver Hohlfeld, and Klaus Wehrle. Is the web ready for http/2 server push? In *Proceedings of the 14th ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2018.

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

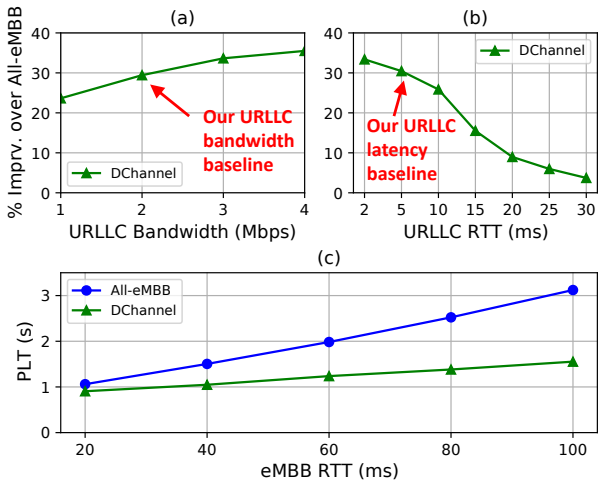

Figure 8: *DChannel under varying eMBB and URLLC network conditions, conducted using the baseline (RTT (ms) / bandwidth (Mbps) for eMBB and URLLC are 40 / 200 and 5 / 2) with a single varying parameter.*

A Appendix

A.1 Algorithm Listing

The packet steering algorithm is listed as Algorithm 1. Note that since HBC bandwidth (B_{hbc}) is typically large and relatively hard to measure, for simplicity, we omitted the queueing delay caused by $(size(P_n) + Q_{hbc}(t_n))/B_{hbc}$.

Algorithm 1: DChannel steering algorithm

```
Result: Send a packet to either HBC or LLC
c_llc = t_now + llc_prop + (size(packet) + queue_llc) /
  band_llc;
c_hbc = t_now + hbc_prop;
rewards = c_hbc - max(prev_c, c_llc);
cost = (size(packet) + queue_llc) / band_llc;
if rewards > alpha * cost then
    send(pkt, LLC);
    prev_c = max(prev_c, c_llc);
else
    send(pkt, HBC);
    prev_c = max(prev_c, c_hbc);
end
```

A.2 Parameter Calibration

The results of our parameter sweep are shown in Table 5.

A.3 Understanding DChannel Performance

Below, we investigated how DChannel performs under tightly controlled network variables. We used a fixed network latency and bandwidth for the experiments below.

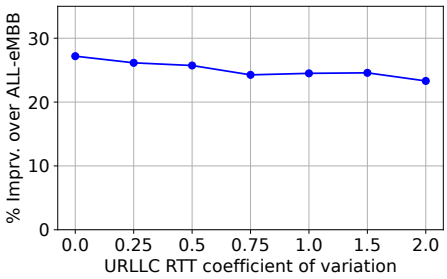

Figure 9: *DChannel PLT improvement under time-varying URLLC RTT randomly generated according to a Gaussian distribution.*

A.3.1 Performance under high eMBB RTT

We evaluated DChannel under eMBB RTT inflation and found it to be resilient towards the RTT increase (Fig. 8c). Specifically, DChannel is 2× faster than the baseline when eMBB RTT is held at 100 ms, which is possible in device mobility as Tab. 1 shows. As eMBB RTT increases, DChannel web PLT degrades at a slower rate compared to All-eMBB because it uses eMBB primarily for bandwidth-sensitive (low rewards) traffic that is not affected as severely by the increased latency. Inline with our trace-based evaluation under mobility, the parallel channel setup can be extremely effective when the eMBB channel quality degrades.

A.3.2 Varying URLLC latency

Although URLLC is expected to deliver consistent low latency, we evaluated latency inflation on URLLC to reflect scenarios where web traffic is de-prioritized in favor of critical traffic. DChannel is still superior even when URLLC latency increases up to 30 ms and eMBB held at 40 ms (Fig. 8b). As expected, as URLLC latency increases and becomes closer to eMBB, the PLT improvement rapidly diminished because LLC no longer offers a competitive resource despite its higher cost.

To evaluate DChannel sensitivity to URLLC latency instability, we changed URLLC propagation latency over time to random values that follow Gaussian distribution. We kept the URLLC mean to 10 ms and ran experiments with an increasing amount of variation, controlled with the Gaussian distribution's coefficient of variation (CoV). We set the URLLC bandwidth to 2 Mbps while the eMBB RTT and bandwidth are 40 ms and 200 Mbps. We found that DChannel performance is relatively robust to the URLLC latency change (Figure 9); the PLT improvement only drops from 27.18% to 23.31% when the URLLC latency CoV changes from 0 to 2.

A.3.3 Varying URLLC bandwidth

The URLLC bandwidth is generally limited to ensure its reliable and low latency service. We tested DChannel under varying URLLC and summarize that its improvement flattens as URLLC bandwidth increases past 2 Mbps (Fig. 8a). As

Table 5: *Percentage of improvement or 'speedup' in PLT (%ps.) along with the percentage of bytes (%sz.) sent via URLLC for various values of the eMBB channel RTT. The table also shows how the different values of* α *affect the performance benefits.*

RTT	α = 0		α = 0.25		α = 0.5		α = 0.75		α = 1		α = 2		α = 3	
(ms)	***%ps.***	***%sz.***	***%ps.***	***%sz.***	***%ps.***	***%sz.***	***%ps.***	***%sz.***	***%ps.***	***%sz.***	***%ps.***	***%sz.***	***%ps.***	***%sz.***
20	16.7	13.3	16.6	10.7	18.4	5.9	18.9	5.7	18.5	5.5	15.6	4.7	14.2	4.1
40	31.1	18.8	33.2	16.2	34.0	13.9	34.7	11.9	34.0	11.1	32.8	8.5	31.8	5.8
60	37.5	22.6	40.9	19.5	41.2	17.4	42.1	14.9	40.8	14.2	40.6	11.1	37.4	9.8
80	43.2	26.3	46.3	22.9	46.3	19.9	46.3	17.7	47.1	16.5	45.6	13.3	45.1	11.6
100	47.1	29.4	48.6	26.1	49.7	22.5	50.2	19.9	50.7	18.3	49.8	14.9	48.5	13.1

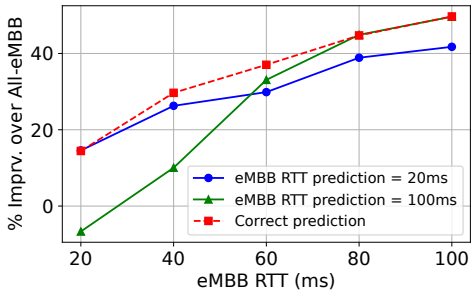

Figure 10: *Effect of underestimating and overestimating eMBB latency on the mean PLT improvement.*

URLLC bandwidth increases, DChannel aggressively offloads packets belonging to a larger transfer (e.g., HTTP response) to URLLC. It may not affect the completion time, however, as it still needs to wait for the remaining part to be transferred over the eMBB.

A.3.4 Working with Incorrect Latency Estimates

DChannel requires estimates of eMBB and URLLC latency to calculate rewards and cost (§3.4). While URLLC latency is predictable, it can be hard to get an accurate measurement of eMBB latency, especially under mobility. To better understand the sensitivity of DChannel to the latency estimates, we evaluated DChannel under underestimated and overestimated latency. Fig. 10 shows that underestimating eMBB latency is safer: When DChannel underestimates eMBB latency as 20 ms (from 100 ms, which is 5× higher), the PLT improvement only decreases by 8%. Underestimating eMBB latency will underestimate the rewards, causing a more conservative use of URLLC and ensuring that the offloaded packets are high rewards packets. Overestimating the latency will, in contrast, overestimate the rewards, resulting in unnecessary packets being offloaded to URLLC, and slowing down the channel. Per Fig. 10, overestimating eMBB latency by 5×, DChannel gets worse performance than the baseline.

A.3.5 DChannel under TCP BBR

Since DChannel steers packets over two channels, the sender may notice an abrupt change in the flow RTT. We evaluated DChannel in TCP BBR, which uses RTT measurement to determine whether a path is congested. Tab. 6 shows the result when low (RTT=20 ms) and high (RTT=100 ms) eMBB latency are used. When the eMBB RTT is 20 ms, BBR works perfectly with DChannel because eMBB and URLLC latencies are not that different. As the latency gap widens, however, BBR starts to treat abrupt latency inflation as a congestion signal, reduces its windows rate, and increases PLT. We found that for 20% of the web page loads, DChannel performs worse than All-eMBB. These sites rely on a single TCP connection to deliver most web objects, and that flow suffers from a low sending rate. This can happen as DChannel accelerates the early packets (such as TCP SYN) to URLLC and suddenly switches back to eMBB once it sees large traffic. The abrupt RTT change gives a wrong congestion signal to the sender. One possible solution is to modify BBR to be aware of eMBB and URLLC use so that it can tolerate a change in RTT and maintain its sending rate. We leave this as future work. Alternatively, we can use different heuristics (pkt-uplink) that will send all uplink packets to URLLC and downlink packets to eMBB. This heuristic is based on the observation that client traffic is generally small and accelerating those packets will accelerate the flow RTT in a more consistent way. Table 6 shows we can get 39% improvement in PLT under BBR from this scheme.

Table 6: *Mean PLT with TCP BBR under different eMBB RTTs. The URLLC RTT is set to* 5 ms

	RTT=20ms	***RTT=100ms***
All-eMBB	915 ms	2661 ms
DChannel	716 ms (21%)	2713 ms (-2%)
pkt-uplink	860 ms (6%)	1628 ms (39%)

A.4 DChannel rewards calculation accuracy

We evaluated DChannel packet rewards (R) calculation accuracy by comparing the calculated rewards and the *real* rewards. As we used a trace-based emulated network, we knew both network bandwidth, latency, and the link's queue depth at any given time. Leveraging this information, we can calculate the

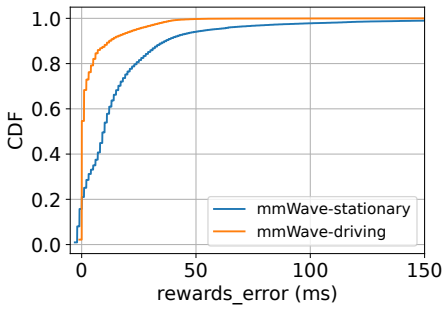

Figure 11: *DChannel calculates rewards error distribution ($R_{real} - R_{est}$) across individual packets. A positive value denotes R underestimation. We limited the x-axis due to the high rewards error in the tail (*155 ms *and* 604 ms *at the 99th and 100th percentile).*

real rewards before we commit a packet to a link. Figure 11 shows the rewards calculation error for the mmWave stationary (MM-S) and driving (MM-D) network traces. DChannel is able to accurately estimate R with just 12 ms of error in the 90th percentile in the stationary network traces due to the network latency being relatively stable and the bandwidth being large. In the driving traces, the error is noticeably higher with a long tail. However, 90% of the time the error is less than 37 ms. The R error mainly arises from the less accurate network eMBB latency estimation and the impact of ignoring the eMBB queueing delay (§A.1), since eMBB bandwidth may be low, and the delay becomes more significant).

The above is also why, as can be seen in the figure, R is rarely overestimated. Underestimation is the preferred direction of error, as we have shown that DChannel can tolerate some incorrect latency estimates and rewards underestimation (§A.3.4). However, better network measurement may improve DChannel performance; we leave this as future work.