M2HO: Mitigating the Adverse Effects of 5G Handovers on TCP

Zhutian Liu[†], Qing Deng[†], Zhaowei Tan[†], Zhiyun Qian[†],
Xinyu Zhang[‡], Ananthram Swami[§], Srikanth V. Krishnamurthy[†]
[†]University of California, Riverside, [‡]University of California, San Diego, [§]DEVCOM Army Research Lab

Abstract

The advent of 5G promises high bandwidth with the introduction of mmWave technology recently, paving the way for throughput-sensitive applications. However, our measurements in commercial 5G networks show that frequent handovers in 5G, due to physical limitations of mmWave cells, introduce significant under-utilization of the available bandwidth. By analyzing 5G link-layer and TCP traces, we uncover that improper interactions between these two layers causes multiple inefficiencies during handovers. To mitigate these, we propose M2HO, a novel device-centric solution that can predict and recognize different stages of a handover and perform state-dependent mitigation to markedly improve throughput. M2HO is transparent to the firmware, base stations, servers, and applications. We implement M2HO and our extensive evaluations validate that it yields significant improvements in TCP throughput with frequent handovers.

CCS Concepts

• **Networks** → **Network protocol design**; **Mobile networks**; **Network mobility**; **Transport protocols**.

1 Introduction

In recent years, 5G cellular networks have seen extensive deployments by mobile carriers. One of the new features promoted by 5G is the use of millimeter wave (mmWave) bands. Operating at frequencies between 24.25 GHz and 71 GHz [16], these bands can theoretically offer data rates of up to 20 Gbps, a remarkable improvement over traditionally used sub-6GHz bands. On a global scale, major carriers are actively deploying 5G mmWave base stations [7, 9–11], and consequently, according to a 2022 measurement, the aggregated throughput with commercial carriers could reach over 3 Gbps [3]. The high bandwidth brings the promise of enabling emerging throughput-intensive applications in 5G, such as Virtual/Augmented Reality (VR/AR) [27, 34], unmanned aerial vehicles (UAV) [58, 62], and autonomous driving [39, 52].

However, the physical characteristics of mmWave links also introduce unique challenges. mmWave can only travel as a direct wave, and is extremely vulnerable to various blockages like walls, humans, and even hands [28, 36]. Consequently, operators have to deploy a large number of mmWave base stations to ensure consistent connectivity, resulting in frequent handovers during mobility [33, 46–48, 60].

In this paper, we attempt to answer the following question: "Would frequent handovers, especially those due to mmWave, impose a negative effect on TCP performance?" We focus on one of the most deployed TCP variants, CUBIC [32]. Research on TCP performance in 5G has shown that TCP may underutilize the available bandwidth [30, 42, 46, 49, 54, 60, 63], but how the interactions between 5G link layer and TCP would incur such deficiencies is yet to be fully understood.

To this end, we first conduct an extensive measurement study under two mobile operators in six locations from three cities (§3). We collect 190 GB of traces at both the TCP and the 5G link layers, and investigate whether TCP CUBIC can realize the potential promise of 5G mmWave and how mobility-induced events impact TCP performance. Our measurements show that, unfortunately, frequent handovers indeed cause TCP to severely underutilize the available bandwidth. In our experiments, TCP only achieves an average throughput of 468.4 Mbps, while the link capacity measured by the saturation tests reaches 993.1 Mbps. After each handover, TCP goes through a slow ramp-up phase of around 6.7 s before it reaches a stable throughput.

Since 5G promises seamless handovers with small disruptions, what causes this significant performance underachievement? To uncover the root cause for this unexplained adverse behavior, we analyze the traces of both transport and 5G link layers. We find that improper interactions between the 5G link and TCP are the main reasons for this underutilized bandwidth. First, packets are dropped due to a link-layer context re-establishment procedure before the handover. This triggers duplicate ACKs and greatly reduces the sending rate. This effect manifests in 18.4% of the handovers, which leads to a 63.3% congestion window (cwnd)

ACM MobiCom '24, November 18–22, 2024, Washington D.C., DC, USA

ACM ISBN 979-8-4007-0489-5/24/11
https://doi.org/10.1145/3636534.3690680

reduction, on average. Second, the packets accumulated during handover cause buffer overflows and packet loss. We see this behavior in 17.7% of the handovers, with an average cwnd drop of 70.4%. Third, TCP is unaware of bandwidth changes and fails to quickly increase cwnd to commensurate the bandwidth. It takes an average of 6.7 s and up to 22.1 s to achieve the maximum throughput.

To address these issues, we propose M2HO, a novel device-centric solution designed to mitigate the negative effects on TCP from handovers (§4). It predicts and recognizes various stages of handover by monitoring lightweight 5G control-plane messages from the baseband processor, using an event-based handover prediction algorithm. Given the handover state, M2HO mitigates the associated deficiencies by means of state-dependent actions. It avoids the buffer overflow before handovers through an intelligent buffer estimation algorithm, suppresses duplicate ACKs after handover without impacting reliable data delivery, and incorporates a fast convergence algorithm leveraging TCP's receive window manipulations.

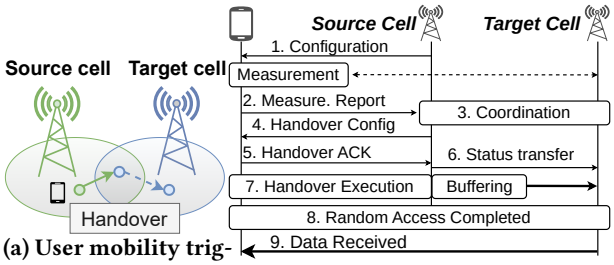
We implement M2HO on a commercial Android device (§5). Notably, M2HO operates as a transparent middle layer on the device side only, requiring no modifications to the firmware, base stations, servers, or applications. Our implementation reads the link-layer events for state transition and alters outbound TCP packets to apply the policies. Since M2HO ideally needs to acquire link-layer messages, which experience delays because of inefficiencies in current firmware APIs (which we cannot fix), we build an emulator to simulate the wireless channel dynamics and deliver the link-layer messages to the device, and leverage user-space APIs to modify the packets. We verify the fidelity of our emulator by comparing real-world and emulated evaluation results.

We set up a testbed and evaluate M2HO by replaying 722 recorded handovers with our emulator (§6). The handover throughput (average throughput from 5 s before to 5 s after a handover) achieved is 314.8 Mbps, which is exceeding that of TCP CUBIC by 45.8%, and is 20.1%-44.8% better than popular alternatives. The convergence time to reach link capacity is reduced to 5.9 ms, which outperforms popular alternatives by 56.3%-68.5%. To achieve the throughput boost, M2HO predicts the handover with 94.8% accuracy. It also recognizes the unnecessary duplicate ACKs caused by handover with 90.3% precision and 98.6% recall. M2HO ensures no packet loss during handover in 85.2% of the cases, whereas with the alternatives losses happen in 44.2-60.3% of the handovers.

2 Background

2.1 5G mmWave

5G has emerged as the new standard wireless technology for wide-area mobile communications. A major facilitator of the promise of high throughput in 5G is the incorporation of FR2 (mmWave) bands into the cellular spectrum. While



(a) User mobility triggers handover. **(b) Detailed handover procedure.**

Figure 1: 5G handover procedure.

LTE and 5G FR1 bands operate at sub-6GHz frequencies, mmWave bands function at markedly higher frequencies from 24.25 to 71 GHz, enabling multi-gigabit bandwidths of up to 20 Gbps. To enable mmWave, 5G incorporates a wide range of enhancements such as massive MIMO, advanced channel coding, and scalable modulation [18].

Both phone vendors and mobile operators are incorporating mmWave, to harness its performance promise and support throughput-intensive applications, such as VR/AR [27, 34]. On the device side, leading phone manufacturers, such as Samsung and Apple, have shipped numerous smartphones with 5G mmWave support [8]. On the network side, major mobile operators are actively deploying mmWave worldwide; examples include Verizon [11], AT&T [9], T-Mobile [7], and China Telecom [10]. In EU, Faroese Telecom and Ericsson are testing major mmWave deployments [5]. Currently, over 100 cities in the US support 5G mmWave technology [6].

In 5G/4G, a device (or user equipment, UE) connects to a cell (called serving cell) for network access[1]. A mmWave cell has limited coverage due to significant propagation loss and vulnerability to blockage and attenuation [47, 49]. Therefore, operators often install a large number of mmWave cells to ensure consistent connectivity as a UE moves. Given the high cost, 5G mmWave is primarily deployed in populated urban locations including malls, airports, and stadiums.

2.2 5G Handover Primer

A UE needs to disassociate from the serving cell (source cell) when it is about to move out of the cell's coverage area. It then associates with a new serving cell (target cell) to maintain connectivity (Figure 1a). This process, called handover, is considered a norm for mmWave, given its limited coverage and dense deployment. Figure 1b shows the detailed handover procedure. It consists of the following major steps.

① The serving cell configures a UE to monitor its serving and neighboring cells' signal strength and the conditions to report. ② The UE periodically measures cell conditions and reports an event once a configured criterion is satisfied (e.g., the serving cell's signal strength is below a threshold). ③

[1]The cell refers to the logical unit providing radio access, while the base station is the physical equipment that manages one or multiple cells. In this paper, we use these two terms interchangeably.

Upon receiving a report, the serving cell decides if a handover is needed. ④ If so, it sends a handover command to the UE. ⑤ The client acknowledges the command and prepares for the switch. ⑥ The serving cell transfers the device status and buffered data to the target cell. ⑦ The UE executes the handover to the target cell; interim, there is no data transfer since it is a hard handover [14]. ⑧ After the switch, the UE starts a random access procedure and establishes a new data path with the target cell. ⑨ Data transmissions are resumed.

Handovers can happen between heterogeneous (mmWave ↔ sub-6GHz) or homogeneous (mmWave→ mmWave / sub-6GHz→sub-6GHz) cells. We call the former *vertical handovers* and the latter *horizontal handovers*. Handovers could be more complicated in reality. For instance, 5G/4G Carrier Aggregation (CA) allows simultaneous connections to multiple cells for higher throughput. In CA, a UE might release and add multiple cells during a handover; nevertheless, it still follows the basic handover procedure in Figure 1b.

2.3 TCP and TCP Congestion Control

TCP is the primary transport layer protocol for reliable data delivery. According to recent research [53, 56], it carries 75%-91.5% of the bytes on the Internet. TCP senders maintain a cwnd for each connection to restrict the in-flight data until a new acknowledgement. The receiver also notifies the sender of its available buffer space as "receive window (rwnd)." The maximum data that the sender can send before expecting an acknowledgment (W_{send}), is determined jointly by cwnd and rwnd, and is chosen so as to not cause congestion or overwhelm the receiver, i.e., $W_{send} = min(cwnd, rwnd)$.

To properly maintain cwnd, a TCP sender needs to infer whether the end-to-end link is under- or over-subscribed (except when the network provides explicit feedback). TCP relies on implicit feedback from the network such as packet loss and delay to perform such tasks. There are two classes of methods used in various TCP versions for such assessments. Loss-based methods take (no) loss as an indicator (TCP New Reno [29], BIC [61], and TCP CUBIC [32]) of congestion, while delay-based methods use packet delay to infer/calculate proper window sizes to avoid congestion (TCP-Vegas [23], Verus [64] and Copa [20]). Some TCP variants take a model based approach and estimate bottleneck bandwidth directly (BBR [26] and TCP-Westwood [44]).

This work is focused on TCP CUBIC [32], the predominant TCP congestion control used today with the largest share of the Alexa Top 20,000 websites [45]. CUBIC has been the default TCP variant for Linux since Kernel v2.6.19 [2] and for Windows OS since Microsoft Windows Server 2019 OS [50]. The findings in CUBIC would also provide insights on behaviors of other loss-based variants with 5G mmWave.

Other TCP congestion control variants, such as Copa [20] and BBR [26], are less widely adopted than CUBIC. Research shows that BBR also encounters issues in highly dynamic 5G network [40, 42], and we will leave the investigation of these variants to our future work.

Similar to other loss-based variants, TCP CUBIC increases cwnd additively upon receiving ACKs, and shrinks it multiplicatively upon experiencing packet loss [51] (as implied by duplicate ACKs (≥3) from the receiver or a timeout [22]). When a packet loss event occurs, TCP CUBIC records the current cwnd as an estimate of the current path capacity, denoted as W_{max}. The cwnd is reduced to $W_{max} \cdot \beta$, where β is a constant multiplicative decrease factor. It then updates cwnd $W(t)$ to be used at time t after the loss event as per a cubic function

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

where C is a constant, and $K = \sqrt[3]{W_{max}(1-\beta)/C}$.

3 Handover Implications on TCP

First, we study whether TCP CUBIC, as the most widely used loss-based TCP variant, can achieve high throughput by exploiting mmWave links, given frequent handovers in 5G due to dense deployments of mmWave cells.

3.1 Methodology

Settings. We conduct an empirical study spanning May 2023 to December 2023, across two major U.S. commercial carriers viz., Verizon and AT&T. The traces are collected from 18 distinct routes (5 indoor routes) at six distinct locations in three cities with mmWave coverage, with each location having at least two mmWave cells. Our experiments cover a total of 33 distinct 5G mmWave cells and 32 sub-6GHz cells. We collect traces under two mobility scenarios: 1) walking at 3 mph; and 2) driving at 20-30 mph causing frequent handovers. For each experimental setting (location, mobility, carrier, etc.), we perform the same test three times.

Experimental Setup. To evaluate TCP performance, we set up a stationary server with an AMD EPYC-Rome Processor and 8GB memory as the TCP sender. It runs Linux 5.15 kernel with TCP CUBIC. It is connected to the Internet through high-speed Ethernet. The Internet speed could reach 2.6 Gbps in our speed test with another PC, thus not being the bottleneck. We use two smartphones as the UEs (TCP receivers): Google Pixel 5 and Samsung Galaxy S22, both equipped with 5G mmWave and LTE capabilities.

Measurement Tools. As our objective is to analyze whether TCP CUBIC over 5G mmWave is able to exploit its full bandwidth and correlate the obscured degradations with potential 5G link-layer behaviors (e.g., handovers), we use multiple measurement tools to acquire data from different layers.

- *iPerf3.* We use iPerf3 to generate TCP traffic and statistics. It is configured to report timestamped bit rate, cwnd, and round-trip-time (RTT) for each 100 ms interval.

- *UDP Flooding.* We develop a UDP flooding app to saturate the wireless link, for measuring the link capacity and isolating TCP artifacts such as cwnd adaptation and retransmissions. The UDP packets are created with incremental sequence numbers to quickly capture packet loss, duplication, and out-of-order packets.
- *tcpdump.* We use tcpdump to capture raw IP packets, enabling fine-grained per-packet analysis of TCP CUBIC. tcpdump offers insights into events such as packet loss, duplicate ACKs, packet reordering, etc.
- *XCAL [4].* We use this a tool to expose 5G link-layer info from Qualcomm-based firmware. We install the XCAL-Smart app on the UE, and XCAL5 on a ACER laptop with AMD Ryzen 7 5800 CPU and 16 GB memory. The laptop connects to UEs via `adb`, and XCAL5 communicates with XCAL-Smart to fetch and decode link-layer logs. We write a Python script to extract handover-related messages (e.g., measurement reports, handover commands).

3.2 TCP Performance under Handovers

Underutilization of Bandwidth. In our experiments, we find that TCP throughput can significantly benefit from mmWave, as expected. On average, the TCP throughput is 468.4 Mbps with mmWave cells, as compared to 121.5 Mbps with just sub-6GHz cells, with a boost of 285.6%.

However, we find that even during low mobility, the "potential" TCP performance is not reached. In UDP flooding tests conducted separately under the same setting as TCP, the UE in mmWave cells achieves an average throughput of 993.1 Mbps. The throughput exceeds 1 Gbps 63.0% of the time, with the maximum being 1.7 Gbps. However, TCP throughput only reaches 468.4 Mbps on average, utilizing 47.2% of the mmWave link capacity, lower than the conservative estimation of 75% of UDP throughput [66].

What causes TCP to under utilize the bandwidth? We hypothesize that frequent handovers could be a key reason. To this end, we perform a TCP throughput test while keeping the UE static and connected to a mmWave cell. In this case, the TCP throughput converges to more than 80% of the link capacity. This indicates that TCP CUBIC is indeed capable of achieving this markedly high throughput in the absence of mobility. We next conduct a more in-depth analysis.

Handovers Negatively Affect TCP Throughput. We analyze the collected XCAL logs and extract handover commands from the `mobilityControlInfo` field in the Radio Resource Control (RRC) messages. These messages are timestamped and are associatable with TCP measurement results. In total, we capture 329 sub-6GHz-sub-6GHz, 109 sub-6GHz-mmWave, 100 mmWave-sub-6GHz, and 184 mmWave-mmWave handovers in our experiments.

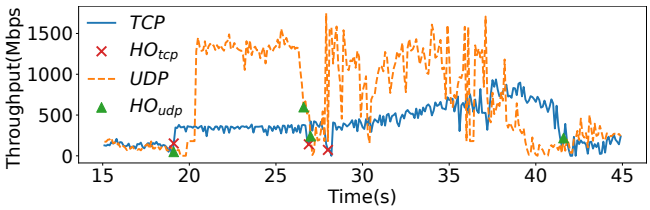


Figure 2: Comparing TCP/UDP throughput in sample traces.

We observe that during mobility, handovers are common. The average time that a UE stays in a sub-6GHz (mmWave) cell before handover is 15.5 s (7.2 s). Handovers happen every 22.1 s on average, in walking tests and every 9.2 s, in driving tests. When walking (driving), 60% (90%) of the connections last less than 20 s before handovers. A device that is in a mmWave cell, in 90% of the cases, will handover to another cell in 25.8 (10.0) s in walking (driving) tests.

We confirm that handovers indeed negatively impact TCP throughput. After a handover, cwnd is reduced by 48.3 % on average, and it takes 6.7 s for TCP to converge to a stable cwnd. The average throughput at 3 s after handover is reduced by 26.3%, on average. In contrast, without a handover, the TCP throughput only fluctuates with small variance. The throughput reduces by more than 50% for only 0.73 times/minute in our experiments. Even if it happens, the throughput ramps up within 1.8 s on average. In addition, there are very few packet losses, with 1.25 losses/min on average, keeping the cwnd stable.

Figure 2 depicts a typical driving trace with TCP and UDP tests under the same setting independently. We plot the TCP throughput and link capacity measured by UDP flooding, with handover events highlighted. At 19 s, a UE handovers to a mmWave cell. The link capacity quickly exceeds 1 Gbps; however, the TCP throughput is still low. Five seconds after this handover, TCP achieves only 33.56% of the link capacity. At 26 and 28 s, two horizontal handovers to other two mmWave cells occur. TCP CUBIC throughput encounters brief drops and recovers to its pre-handover value. Subsequently, TCP gradually reaches its peak throughput in 10 seconds at around 38 s, which still fails to converge to the high mmWave available throughput. TCP throughput only achieves 50.62% of the link capacity on average during the entire connection duration across the mmWave cells.

3.3 Deficiencies of CUBIC under Handovers

Why do handovers impose such a negative impact on TCP throughput? To answer, we delve into the traces and associate cellular-specific events with transport layer behaviors.

Finding 1: Handover disruption time is NOT an issue. Although 3GPP rel-16 proposes soft handovers in 5G [12], our measurements indicate that only hard handovers are used. During handover execution, no data is transferred. One

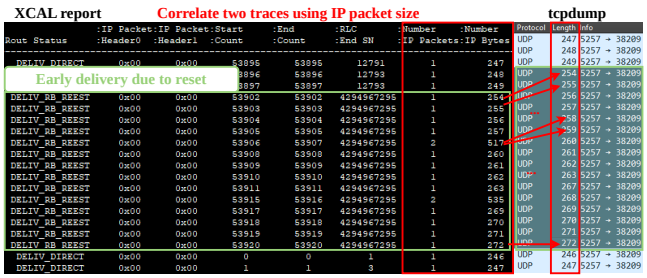


Figure 3: XCAL report and tcpdump packet trace side by side.

hypothesis is that such disruptions trigger timeouts (RTO) on TCP, causing cwnd to reduce drastically.

However, we find that each such disruption is small and will not cause any performance degradation. The average disruption time is only 27.0 ms. Combined with the average RTT of 69.7 ms in our measurement, the overall latency is still less than the default minimum TCP RTT of 200 ms on Linux [1]. We further confirm from our tcpdump traces that among all 722 handovers, only 11 (1.5%) experience timeout, and all of them are caused by random access failures.

Finding 2: Handover causes packet losses at the source cell. We find that improper interactions between 5G and TCP during handover, rather than disruptions, cause the performance degradation. First, TCP clients experience packet losses in the source cell. After handover execution, a client receives out-of-sequence packets, indicating that certain packets are discarded by the source cell and are not forwarded to the target cell. This results in multiple duplicate ACKs to the server, stimulating congestion events and cwnd reduction. In our traces, we see such duplicate ACKs in 18.4% of handovers. They result in a 63.3% cwnd reduction, on average.

Why would this happen given that the UE link layer is supposed to buffer out-of-order packets and wait for retransmission? We uncover that, UE releases the link-layer context (including info such as link-layer sequence number) with the source cell after receiving the handover command [15]. As a result, the UE link layer releases the buffered data to the upper layer, regardless of whether they are in order or not. Therefore, TCP observes the packets with sequence number gaps. We confirm this behavior using XCAL. Although the context release action cannot be directly captured, we indirectly verify it by observing a sequence of packets labeled as "delivered to upper layer due to context release." Besides, packets after handover are assigned new link layer sequence numbers, indicating that a new context is established.

We further validate that the gaps in the transport layer are indeed caused by context release. This requires mapping cellular link-layer packets (collected by XCAL) to TCP packets (collected by tcpdump), which is infeasible in our experimental setting. We thus design an experiment, where the server sends UDP packets with sequentially increasing packet size.

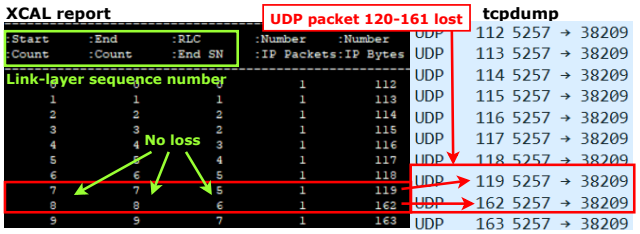


Figure 4: Start/End count in XCAL report indicates the sequence numbering of link-layer packets.

This is used to correlate the (missing) packets between tcpdump and XCAL using packet size as the identifier. Figure 3 shows such an example. From XCAL, link-layer packets of 254 to 272 bytes are delivered to the upper layer due to context release, while packets of 250 to 253 are missing. They match the pattern observed in the tcpdump trace.

Finding 3: Handovers cause buffer overflows in target cells. In addition to packet losses at source cells, we also discover packet losses after handovers at the target cells. Shortly after handover execution, a target cell experiences packet loss in 17.7% of all the recorded handovers. It is more severe when the source cell is mmWave, occurring in 68.7% of the cases. The number of lost packets, ranges from 1 to 995, leading to a 70.4% cwnd reduction, on average.

To understand where this loss stems from, we examine XCAL traces using UDP packets with incremental sizes again, and find that link layer does not discard any packets. Figure 4 shows a typical XCAL trace post-handover, where the link-layer sequence numbers are in order while the transport layer observes losses (lines 120-161 in the figure). The device receives link-layer packets with in-order sequence numbers. This implies that these lost TCP packets are already discarded before the target cell starts to serve the user.

We infer that, these packets are lost at the target cell due to a buffer overflow. During a disruption from a handover, both pending packets from the source cell and new packets from the sender cannot be sent to the UE. These are accumulated in the target cell buffer. The TCP sender, unaware of the ongoing handover, sends data until the send window is full, often causing overflows of this buffer. We cannot directly observe such drops from device-side logs, but indirectly validate this hypothesis. First, we observe that only new data sent after the handover commencement, are lost. This means that the lost packets are those at the end of the queue, likely due to buffer overflow. Second, buffer overflow explains why the loss is more likely when the source cell is mmWave. The sender is more likely to have a larger cwnd when the client is in a mmWave source cell, causing more data accumulation in the target cell queue buffer upon handover.

Finding 4: Slow window growth after handover. Finally, we show that the congestion window grows at a very

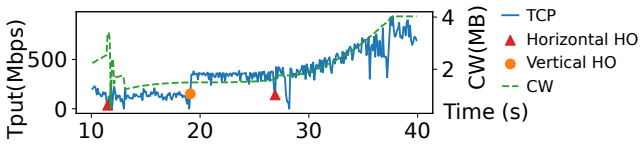


Figure 5: TCP cwnd slow ramp up after handover to mmWave.

slow rate even if the link capacity is greatly increased after vertical handovers to mmWave cells. For all types of handovers, it takes 6.7 s on average for cwnd to converge[2]. This issue is especially serious for vertical handovers from sub-6GHz to mmWave cells, where the available bandwidth increases greatly. It takes 17.2 s on average for TCP throughput to converge to the available bandwidth. Considering that a device often stays in a cell for less than 20 s before handover (§3.3), throughput barely converges in our experiments.

Figure 5 shows a typical trace of cwnd and throughput changes. There are three handovers during 30 s of mobility. At 11.5 s, a horizontal sub-6GHz handover causes consecutive packet losses and cwnd reduction. At 19 s, a vertical handover to mmWave increases the throughput to 384.4 Mbps. Another horizontal handover between mmWave cells at 27 s increases the cwnd at a slow pace, over an extended 24.5 s to raise the throughput to the link capacity. Note that although TCP converges to the pre-handover throughput, it fails to reach the high mmWave capacity, causing severe under-utilization.

The root cause is TCP CUBIC's loss-based congestion control algorithms' conservative cwnd increase during congestion avoidance. Specifically, as per Eqn. 1, TCP CUBIC probes the Bandwidth-delay product (BDP) at a slow pace when cwnd is around W_{max}. As shown in Figure 5, there is a 12 s slow probing phase around W_{max}. CUBIC designs this slow probing phase with the assumption that W_{max} is close to the actual channel capacity. However, W_{max} is not proactively estimated to reflect the increased BDP, after a vertical handover to mmWave. Instead, it is only updated using the cwnd when congestion (packet loss) is conceived to occur. In this case, W_{max} is set to the cwnd before handover, failing to reflect the bandwidth of the new channel capacity.

3.4 Design Insights for Mitigation

Cross-Layer Solution across Link and Transport Layers. Our findings highlight the need for a cross-layer solution that seamlessly integrates information from both of these layers to mitigate improper interactions. The 5G link layer will provide handover-related and real-time capacity info, while TCP will adapt its behavior based on link-layer information.

Handover State-Dependent Solution. As our investigation has uncovered, the causes of TCP underperformance exist in distinct stages of handovers, both before and after the execution. For proper mitigation, we need to design state-dependent actions. A prerequisite to such a solution is to identify handovers in an accurate and timely way. This way, proper countermeasures can be taken before loss occurs.

[2]Here we consider TCP CUBIC throughput to converge when it reaches 75% of UDP throughput, which is its limit in static case due to intrinsic overhead as shown in a large-scale measurement test [66].

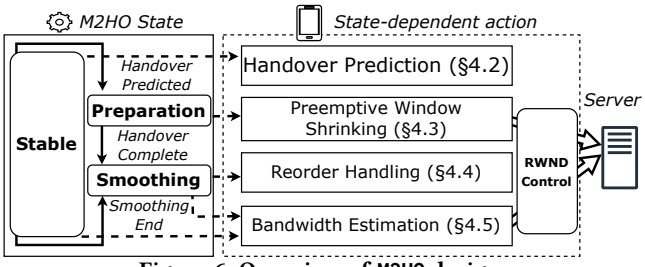


Figure 6: Overview of M2HO design.

4 Mitigating Handover Effects with M2HO

4.1 Overview

We introduce M2HO, a novel solution designed to mitigate all mentioned negative effects on TCP, caused by handovers. M2HO is standards-compliant and operates without modifying the 5G protocol stack. As a cross-layer solution, M2HO analyzes the link-layer signaling messages and predicts impending handovers. It subsequently masks their adverse effects via device-side packet manipulation mechanisms. M2HO leverages link-layer information to infer the available bandwidth, and implicitly adjusts the sending rate using the TCP rwnd. M2HO is a pure device-side solution, requiring no infrastructure assistance or specification modifications. Since M2HO significantly mitigates packet loss, it is applicable to any loss-based TCP variant. Even though the uncovered deficiencies have lower impact on delay-based TCP variants, M2HO can also help them by reducing the required retransmissions.

The core of M2HO is a state machine with three states: 1) Stable state, when no pending handover exists; 2) Preparation state, when a handover is predicted but prior to its execution; and 3) Smoothing state, after handover execution. After the packets buffered before handover are delivered, the state machine goes back to the stable state. Our action space includes four state-dependent actions, as shown in Figure 6.

- *Stable State: Lightweight, Event-Based Handover Prediction (§4.2).* In the absence of a pending handover, M2HO monitors the control-plane signaling messages at the link layer infrequently (once every few seconds), and predicts any upcoming handovers early with high accuracy. It leverages the insight that a handover decision is event-based and the device could infer the cell's decision by locally checking events in measurement reports.
- *Preparation state: Preemptive Window Adjustment to Mitigate Packet Drops (§4.3).* M2HO tackles potential drops caused by cell buffer over-subscription by preemptively adjusting the rwnd to reduce server's sending rate, without direct server modifications. M2HO incorporates a novel

Table 1: 5G measurement events.

Event	Explanation
A1	Serving cell becomes better than threshold
A2	Serving cell becomes worse than threshold
A3	Neighbor becomes offset better than serving cell
A4	Neighbor cell becomes better than threshold
A5	Serving cell becomes worse than threshold1 AND Neighbor becomes better than threshold2
B1	Inter RAT neighbor cell becomes better than threshold
B2	Serving cell becomes worse than threshold1 AND Inter RAT Neighbor cell becomes better than threshold2

algorithm to estimate buffer size leveraging 5G/4G cells' burst-based transmission scheme [21].

- *Smoothing State: Suppressing Unnecessary Congestion Indicators (§4.4).* M2HO identifies duplicate ACKs caused by source cell packet losses as *unnecessary* as they will be delivered via a second attempt by the target cell. This insight allows M2HO to suppress the duplicate ACKs after handover execution to avoid cwnd reduction. M2HO intelligently recognizes the end of such a secondary delivery and releases the suppression.
- *Stable State: rwnd-Based Sending Rate Control and Available Bandwidth Estimation (§4.5).* To ensure a fast cwnd convergence after an abrupt bandwidth increase, M2HO opts to control the sending rate using rwnd after a vertical handover to a sub-6GHz cell. M2HO dynamically estimates the available bandwidth and updates rwnd to avoid a following loss event and cwnd reduction. When the UE hands over back to a mmWave cell, M2HO releases rwnd control and the sender will still enjoy a large cwnd.

4.2 Event-Based Handover Prediction

The first critical component of M2HO is to predict an imminent handover, which aids the UE in making informed preemptive actions before the actual handover execution. A simple solution without prediction is to react upon receiving a Handover Command. However, our measurements indicate an average interval of 32.7 ms between the reception of the handover command and the actual execution. This short interval is insufficient for triggering changes in TCP server actions to prevent the adverse effects.

Prediction Algorithm. M2HO predicts a handover before the arrival of Handover Command with high accuracy by analyzing lightweight 5G link-layer messages. The key insight is that the 5G handover is *interactive and event-driven*. As discussed in §2.2, a UE sends a measurement report if the measured signal strength meets the criteria of one of the configured events. The list of possible events are standardized [13], and a subset of them is shown in Table 1. An example is A2 Event, where a neighbor cell's signal quality is higher than a threshold pre-configured by source cell.

Algorithm 1 Handover prediction algorithm.

```
MR ← the event in a measurement report object.
s, m ← serving cell and a measured cell as in the event.
if MR = A3 and s.type = m.type then return HO
if s is sub-6GHz then
    if MR = B1 then return HO
    else if MR = A5 and m is mmWave then return HO
else if s is mmWave and MR = A2 then return HO
else Return No HO
```

While the operators can configure events with different thresholds for each cell, they usually employ the same handover-triggering logic based on the reported events [55]. Therefore, in M2HO, the UE captures such events from measurement reports from the 5G link layer locally and predicts a handover by inferring the cells' decision logic. We design a lightweight prediction algorithm, depicted in Algorithm 1. We observe that when the serving cell is sub-6GHz, the serving cell will initiate a vertical handover to a mmWave cell whenever feasible. This includes the B1 event, when a mmWave cell has a signal better than the B1 threshold, and the A5 event when a mmWave cell has a better signal strength than an A5 threshold and the current cell's signal strength is below another A5 threshold. On the other hand, if the serving cell is a mmWave cell, a vertical handover to the sub-6GHz cell is only initiated upon an A2 event, which indicates that the signal strength of the serving mmWave cell is no longer acceptable. Meanwhile, a horizontal handover is initiated when a neighbor cell's signal strength becomes better than that of the serving cell by an offset (A3). While there are other events defined by 3GPP (e.g., A4 and B2), they are not directly correlated with handover from our observations and thus excluded from M2HO's decision making.

M2HO scrutinizes the RRC messages and extracts reported events from measurement reports. If any handover-triggering condition is met, it predicts an upcoming handover. To mitigate false positives, M2HO employs a fallback mechanism, canceling predicted handover actions if no handover occurs. M2HO includes two conditions for such fallbacks: 1) An A1 event is generated after prediction, since A1 indicates a good signal quality from the serving cell. This is usually perceived as a signal that the handover will be annulled since the signal strength of the serving cell is improving. 2) A handover command is not received within $2 \cdot T_2$ after prediction, where T_2 is the average interval between the measurement event and handover command as recorded by M2HO. This indicates a mis-prediction and a handover is unlikely to happen.

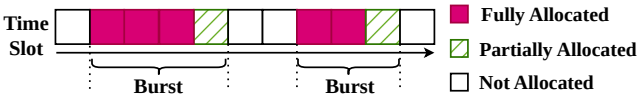


Figure 7: Identify the burst with fully allocated slots.

4.3 Preemptive Window Shrinking

After a handover is predicted by M2HO, it enters the preparation state, where the UE aims to mitigate the potential overflow after handover execution as discussed in §2.2. Without access to the cell's internal buffer, M2HO opts to preemptively prevent the TCP server from sending excessive data.

Controlling the server's sending rate via receive window tuning at the UE. M2HO is a client-side-only solution that avoids any server-side changes. As the sending rate is bounded by $rwnd$, to prevent the sender from overflowing the receiver's buffer, M2HO encapsulates the intended sending rate, $rwnd_{est}$, as rwnd in all uplink packets. $rwnd_{est}$ is supposed to be a small value, and once the first packet with the modified rwnd arrives at the server, it refrains from sending more data. The volume of in-flight packets will converge to $rwnd_{est}$, avoiding packet losses from buffer overflow.

There is a corner case where the first packet with modified rwnd arrives at the server too late, causing in-flight packet numbers to still exceed the buffer size. However, M2HO still performs no worse than vanilla TCP CUBIC. Importantly, a reduced rwnd will not further increase the in-flight data and aggravate buffer overflow. Besides, if packet loss indeed occurs, the sending rate after handover will be decided by cwnd and M2HO will not impact throughput post-handover. We will also show in §6 that a late rwnd barely happens thanks to our early handover prediction algorithm.

Estimating buffer size. A key design choice is to properly select $rwnd_{est}$ for reasonable sending rate control. A straightforward solution is to set $rwnd_{est}$ as 0, which fully stops the server from any further data transmission, as in M-TCP [24]. However, this wastes the available bandwidth between the measurement report and handover execution times, which is 178.9 ms on average, and can be up to 310.2 ms. Such a long disruption might even affect application services.

We note that M2HO needs to make sure that the data volume in-flight is smaller than the max buffer size of the target cell, $s_{in-flight} \leq s_{max}$, as the buffered data will be no more than in-flight data. Consequently, an oracle M2HO would select $rwnd_{est} = s_{max}$, since the $s_{in-flight}$ is bounded by $rwnd$.

However, the max buffer size, s_{max}, is unknown to UE. We estimate this value by leveraging a key insight from cellular downlink scheduling [21]. To serve all connected UEs, a cell will clear each UE's buffer before serving another one. This scheme is shown in Figure 7. As background, 5G splits the time domain into time slots. In each time slot, resources on frequency domains are split into Resource Blocks (RBs). When the cell schedules downlink data for a UE, it will assign most RBs to it in consecutive time slots until the buffer is cleared. We call this period a burst period. The beginning of the burst is marked by a time slot when RBs are fully allocated to the user. The burst ends with a time slot with partially allocated RBs followed by one with none.[3]

Since the burst drains the current buffer data, the UE could count the data received within one burst period to estimate the data in buffer before the burst, effectively getting a lower bound of s_b, on s_{max}. M2HO keeps counting the data in each burst period, and updates the lower bound on the buffer if the new estimate is larger than the previous estimate, denoted as $\hat{s}_{max}$. Eventually, we have $s_b \leq rwnd_{est} = \hat{s}_{max} \leq s_{max}$. M2HO updates $\hat{s}_{max}$ for current cell in every burst.

M2HO needs to estimate the buffer size of the target cell. When using bursts as above, it only learns the buffer size of the serving cell. Thus, M2HO also records the cell ID and stores its buffer size in the local database. When a handover happens, M2HO reads the target cell ID and gets the buffer size estimate from the history. If the target cell has not been accessed before, M2HO reads the target cell "type" from the handover command and uses the minimum buffer size of the same cell type in the history as a conservative estimate.

Confirming the scheduling algorithm and burst periods in 5G. We cross-validate the scheduling method as claimed in [21] by analyzing our collected traces. For this purpose, we calculate the percentage of the time slots where resources are assigned to the UE belonging to a burst. In our 5G sub-6GHz measurement, the slot length is 0.5 ms and the maximum number of RBs is 106 as per the RRC configuration. We find that 96.5% of the received resources belong to a burst. This confirms the validity of the buffer-clearing scheduling.

However, we cannot verify this pattern in mmWave cells. Due to directional beams, when a mmWave cell allocates RBs to a UE, the radio resource cannot be shared among other UE. Thus, we cannot easily identify the partial slots from resource allocation. Therefore, M2HO assumes mmWave cell's buffer is no smaller than a sub-6GHz's buffer. This is because we observe no buffer overflow behavior in handovers between mmWave cells, while mmWave-sub-6GHz handovers usually suffer from packet losses. M2HO in turn uses the minimum buffer size of the sub-6GHz cells as a safe estimate.

4.4 Handling Packet Reordering

M2HO confirms handover execution completion by finding random access procedure messages at the link layer, and subsequently enters the smoothing state. In this stage, duplicate ACKs are sent from the UE, resulting in the reduction of sender's cwnd. This is unnecessary, as the missing packets will be retransmitted by the target cell. M2HO suppresses such

[3]Note that, we define a time slot as "fully" assigned if more than 90% of the RBs are assigned to the UE, and partially assigned if less than 40% of the RBs are assigned to the UE. This is consistent with the definition in [21].

duplicate ACKs in the smoothing state. It specifies a short time period after handover, during which all outgoing ACKs are discarded. This will ensure that the duplicate ACKs are not seen by the server, not affecting cwnd on the server side.

M2HO intelligently decides when to stop the suppression. A premature termination will potentially incur duplicate ACKs, while a late release might even cause a timeout on the sender side. M2HO releases the suppression when all the data from the source base station have been sent. It uses two indicators. First, if M2HO sees an outgoing ACK with a larger sequence number (SN) than the largest SN before handover, all data tunneled to the new cell has been delivered. Second, the suppression ends when M2HO observes the end of the first burst period. Similar to our observation in §4.3, the target cell will quickly clear the buffer, including data forwarded from the source base station, in a burst pattern.

We note that, M2HO does not adversely ignore legitimate duplicate ACKs caused by actual packet loss. Actual packet loss is potentially triggered by buffer overflow. However, we note that, when such a packet loss event happens, those dropped packets will have higher sequence numbers than the ones received before handover. This already satisfies our first suppression releasing condition, and M2HO will let TCP report duplicate ACKs and trigger a sender-side retransmission.

4.5 rwnd Control w/ Bandwidth Estimation

Traverse the bandwidth gap by maintaining large cwnd. On exiting the smoothing state, M2HO enters a stable state which aims to update the server's sending rate to fully utilize the bandwidth. We note that, it is challenging to increase the sending rate on the device side. Using rwnd is effective in limiting the sending rate, but not increasing it.

M2HO's core idea is to keep sender's cwnd at the level of a mmWave cell even when the UE hands over to a sub-6GHz cell. Thus, when the UE switches back to a mmWave cell, the sender still has a large cwnd. Interim, when the UE is in a sub-6GHz cell(s), M2HO manipulates rwnd to limit the sending rate, preventing loss events from decreasing the cwnd.

Estimating available bandwidth. As the sending rate in sub-6GHz cells is throttled by rwnd, it cannot be set arbitrarily but needs to faithfully reflect the channel capacity. An underestimated rwnd leads to an underutilized link, while overestimation causes packet losses. To this end, M2HO estimates the capacity and adjusts the rwnd continuously. We adapt a simple, client-side estimation algorithm based on RTT variation and real-time bandwidth estimation [38]. M2HO performs RTT estimation using the TCP timestamp option [35]. In every RTT, M2HO counts the bytes received as a conservative bandwidth estimate c_{est}. rwnd is computed as,

$$rwnd = \lambda \times c_{est} \times \frac{RTT_{min}}{RTT} \quad (2)$$

where RTT_{min} is the minimum RTT value measured by M2HO. λ is a tunable parameter that determines the aggressiveness of probing. In our setting, aggressive probing might lead to overestimation of the link capacity, causing packet losses and cwnd reduction. Therefore, we set $\lambda = 2$, a less aggressive value based on the prior cellular experimental results [38].

5 Implementation

5.1 Implementing M2HO

We implement M2HO as a pure user-space daemon on the device side, as shown in Figure 8. It is based on the Android 14 Kernel with Google Tensor and Qualcomm chipsets. The same implementation logic could also be applied to other OSes (e.g., iOS) and chipsets (e.g., MediaTek). All components are written in C with 1,028 lines of code.

Our implementation consists of three major components.

Controller. The *Controller* module maintains the states for M2HO and controls the TCP behavior based on its state-dependent design. With respect to the former, M2HO maintains a local state machine. It interacts with *Event Capturer* to acquire the measurement events and runs event-based handover prediction (i.e., Algorithm 1). For the latter, M2HO implements the burst-based throughput estimation, updating the intended rwnd and sending it to the *Packet Processor* to enforce the packet processing policy. It also sends requests to *Packet Processor* to start and release reorder suppression.

We build a shared memory object for *Controller* to update policy to *Packet Processor*. It supports two commands: "rwnd Update," where *Controller* specifies the new rwnd window (negative meaning release), and "ACK Suppression" with 1 meaning activation and 0 meaning release. To retrieve information from link and TCP layers, we build two shared memory objects, one for packet stats report from *Packet Processor* and the other for link-layer info from *Event Capturer*.

Packet Processor. This module is M2HO's interface to manipulate TCP packets. It is implemented as a *netfilter_queue* process to control TCP packets in user space[4]. At the start of a TCP connection, *Packet Processor* sets *iptables* rules to redirect all TCP packets to the Netfilter queue. When any TCP packet passes through the *iptables*, the module will receive a callback that allows it to modify, release, or suppress the packet. *Packet Processor* changes its strategy upon receiving commands from *Controller* as introduced above. It further logs the sequence number and timestamp for inbound TCP packets and sends them to *Controller*.

Event Capturer. This module monitors cellular information and sends important events to *Controller*. The cellular information in the firmware can be exported through the diagnostic API embedded in the chipset firmware (e.g., Android

[4]If more critical performance is needed, it can be implemented in the kernel.

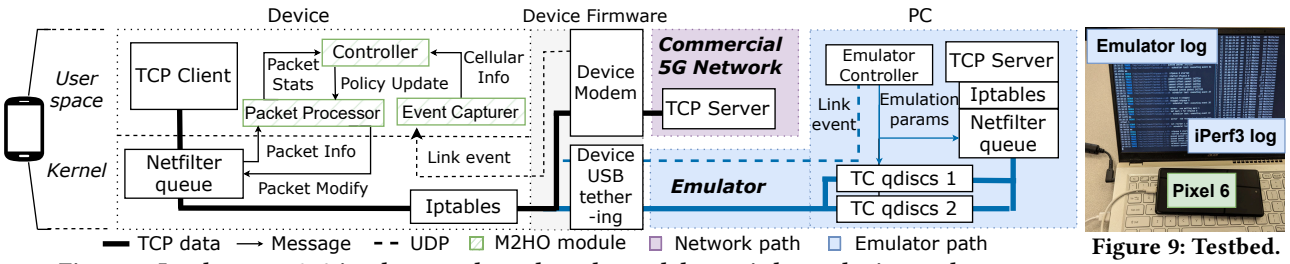


Figure 8: Implement M2HO in phone and emulate channel dynamic by replaying real trace.

Figure 9: Testbed.

dev/diag Port [41]). This module leverages that information to report four events to *Controller*: 1) measurement reports; 2) handover command; 3) random access completion after handover; 4) RB allocation status per slot. The first three events are used for state management, and the last is reported periodically for buffer estimation.

5.2 Implementation-Compatible Emulator

While M2HO implementation is generally applicable, we currently cannot achieve accurate evaluation results as XCAL delivers messages from firmware with significant delays (around 2.4 s). Implementing *Event Capturer* in firmware fundamentally addresses the issue, but we have no access to 5G firmware on commercial smartphones. We are working with industry collaborators to explore the possibility.

To this end, we further develop a trace-driven emulator for evaluation purposes. The architecture is shown in Figure 8. Instead of connecting the phone with M2HO to commercial 5G, we connect it to the PC using USB network tethering[5]. The program on the PC modifies the capacity and emulates handover behaviors, both based on replaying the traces collected during our measurement study. This way, the emulator is transparent to the device and M2HO components. From the phone's perspective, it is as if it is directly running under a cellular network. It has three major functionalities.

Emulating the channel dynamics. The emulator controls the USB data rate to reflect the throughput dynamics. We create two network interfaces on the PC to emulate two cells, applying TBF qdisc and NETEM qdisc to control the link bandwidth and latency, respectively. The configurations of both qdiscs are updated by our emulator controller, which gets bandwidth, average delay, and jitters from our measurements. A Netfilter queue is created between the TCP server and the interfaces to emulate the buffer. All TCP packets have a Netfilter mark that routes them to one of the interfaces.

Emulating handovers. Emulator Controller acquires handover events and measurement reports from the XCAL logs, and then processes the TCP traces to associate handovers with the packet loss events caused by re-establishment. It duplicates and drops the packets in the server Netfilter queue to recreate false congestion behavior. The delay is emulated by holding packets in the queue. Emulator controller will then create the bandwidth configuration of the target cell on an idle interface. After handover execution, it will change the Netfilter marks, so that the future packets will be routed to the new interface that emulates the target cell.

Forwarding link-layer message. The Emulator Controller reads the XCAL logs and forwards (using UDP) the RRC messages, random access messages and RB allocation report, according to the timestamps, to *Event Capturer*. *Event Capturer* reads the messages from the UDP port instead of from the firmware diagnostic port.

Confirming the authenticity of the emulator. To validate that our emulator approximates real cells' behaviors, we disable the M2HO components and run the emulator by replaying all our TCP traces in six locations with 722 handovers. We compare the achieved throughput through the emulated network and under real measurement.

As shown in Figure 12, the measured throughput closely aligns with real-world data. The average throughput difference in each 1s sliding window is 6.26%. We thus consider the emulator to reliably mimic real-world conditions [59].

6 Evaluation

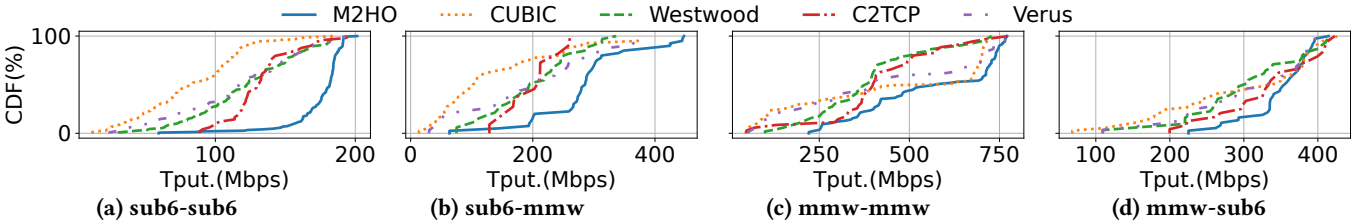
6.1 Evaluation Setup

Testbed. We implement M2HO on a Pixel 6 phone and host the emulator on a laptop running the Linux 6.5 kernel. The laptop is equipped with an AMD Ryzen 7 5800U processor with 16 GB memory. A high-speed USB4 cable connects the two machines. The TCP sender and receiver run on the PC and phone, respectively. The setup is shown in Figure 9.

Setting. We replayed a subset of the original traces spanning 6 hrs, with 722 handovers. They include 513 horizontal and 209 vertical handovers. We process these, and extract the necessary network metrics for the emulator (see §5.2).

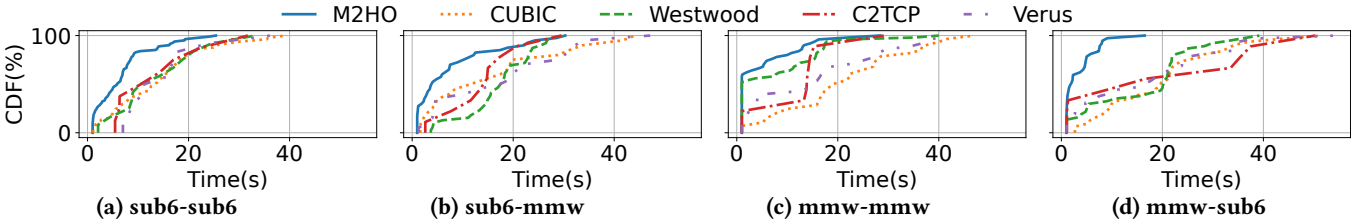
Benchmarks. We apply M2HO over vanilla TCP CUBIC. In addition to CUBIC, we compare the performance of M2HO with other relevant TCP variants, optimized for cellular networks. TCP Westwood [44] is designed for wireless networks with channel bandwidth monitoring using the rate at which ACKs are returned. Verus [65] can adjust the sending rates by

[5]This link can support 3.8 Gbps with a 1.8 ms latency on average, and thus is not a bottleneck in our emulation.



(a) sub6-sub6 **(b) sub6-mmw** **(c) mmw-mmw** **(d) mmw-sub6**

Figure 10: Average throughput in [-5,5] seconds of handover.



(a) sub6-sub6 **(b) sub6-mmw** **(c) mmw-mmw** **(d) mmw-sub6**

Figure 11: Converge time to stable throughput.

HO Type	M2HO		CUBIC				Westwood				C2TCP				Verus			
	Tput.	t	Tput.	η_0%	t	η_1%	Tput.	η_0%	t	η_1%	Tput.	η_0%	t	η_1%	Tput.	η_0%	t	η_1%
sub6-sub6	175.0	6.8	83.2	110	14.3	52	122.4	43	13.7	51	132.7	32	14.3	52	116.1	51	15.3	56
sub6-mmw	287.4	7.7	138.2	108	15.9	52	200.7	43	17.3	56	201.2	43	14.7	48	200.2	44	17.6	56
mmw-mmw	557.8	4.9	447.4	25	21.2	76	387.9	44	7.8	37	419.8	33	12.9	62	430.2	30	15.7	69
mmw-sub6	348.8	3.6	297.7	17	19.9	82	304.1	15	16.7	79	313.9	10	20.7	83	309.1	13	17.0	79
Overall	314.8	5.9	215.8	46	18.7	68	217.4	45	13.5	56	264.1	20	15.7	62	224.6	40	16.5	62

Table 2: Details of throughput (Mbps) and convergence time t(s). $\eta_0(\%)$ and $\eta_1(\%)$ shows M2HO's improvement over other algorithms, while $\eta_0 = (\texttt{M2HO} - algo)/algo$ and $\eta_1 = (algo - \texttt{M2HO})/algo$.

observing cellular network delay. C2TCP [17] is an end-to-end algorithm to accommodate different applications' QoS requirements. For each benchmark, we apply the parameters used in the original papers (Verus, C2TCP) or use the standard Linux implementation (Westwood, CUBIC). For instance, we configure Verus with a 5 ms epoch, a 1 ms decrement parameter, and a 2 ms increment parameter [65]. Note that these baseline variants handle handover implicitly and cannot leverage the additional information from M2HO.

To evaluate M2HO's handover prediction, we compare with two baselines: LTE-VR [55], a handover prediction algorithm for 4G LTE, and HO-Naive, a naive method that treats *any* measurement report as an indication of handover.

6.2 Overall Performance Evaluation

6.2.1 Average Handover Throughput We assess how M2HO improves TCP throughput during handover. The throughput is calculated as the average TCP throughput within the time range of 5 seconds before and after the handover execution. The results are shown in Figure 10. Compared to TCP CUBIC, M2HO achieves 45.8% higher throughput (215.8 Mbps → 314.8 Mbps) across all handovers. In addition, M2HO effectively eliminates the long tail caused by a handover. The 95th percentile of the throughput is improved from 31.9 Mbps to 158.1 Mbps, a 396% improvement.

M2HO also provides a marked improvement over the baseline alternatives: 44.8% over TCP Westwood, 40.1% over Verus and 20.0% over C2TCP. The benefit over C2TCP is not as notable; however, M2HO requires device-side modification only, while C2TCP is an end-to-end solution that requires server side changes that are hard to realize in practice. Besides, C2TCP requires manually adding oracle knowledge of cell configuration parameters such as link delay. This does not work in real deployments as the UE does not have this knowledge. We remove this oracle information and let C2TCP naively set target delay as the average of the two links' (52 ms). This version of C2TCP only achieves 205.8 Mbps on average, and M2HO provides 53.0% higher throughput.

We also evaluate the improvements across different handover types in Table 2. The improvements are most significant for sub-6GHz-mmWave handovers. M2HO achieves a 108.3% throughput improvement over CUBIC and a 43.3% on average, over the other approaches. This is because M2HO can effectively exploit the large mmWave bandwidth after a handover, thus achieving significant throughput boosts. On the other hand, M2HO brings more modest benefits in mmWave-sub-6GHz handovers with a 17.1% improvement over CUBIC and a 12.7% over the other approaches. This is due to the dominant high throughputs in mmWave cells before handover, making improvements on the sub-6GHz link less noteworthy. Here, if we compute the throughput improvement within 5 seconds after handover only, M2HO increases the throughput significantly, with 27.53%, 29.55%, 30.65%, and 31.13% over CUBIC, Westwood, C2TCP, and Verus, respectively.

6.2.2 Throughput Convergence Speed Next, we evaluate the time for TCP to reach the converged throughput after handover. The results are shown in Figure 11. M2HO takes a much

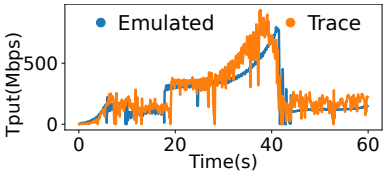


Figure 12: Sample replay.

	sub6-sub6	sub6-mmW	mmW-mmW	mmW-sub6	Total
Precis.	89.5	95.8	98.0	72.5	90.3
Recall	97.5	99.3	100	98.9	98.6

Table 3: Duplicate handling result.

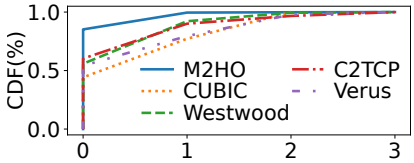


Figure 13: # of losses.

	sub6-sub6		sub6-mmW		mmW-mmW		mmW-sub6		Total	
	p	r	p	r	p	r	p	r	p	r
Naive	66.3	37.9	24.3	28.8	34.3	36.6	20.4	32.8	41.0	39.0
LTE-VR	98.2	60.0	87.2	56.1	88.8	70.2	68.3	40.2	93.1	64.4
M2HO	98.9	93.3	91.9	93.2	88.8	86.1	88.4	59.3	94.8	87.8

Table 4: Handover prediction precision p and recall r (%).

	M2HO	No Pred.
sub6-sub6	128.7	43.8
sub6-mmW	233.7	144.3
mmW-mmW	159.7	80.6
mmW-sub6	146.8	65.8

Table 5: Earliness (ms) comparison.

shorter time to converge in all types of handovers. The average time with M2HO is 5.9 s, which is a 68.32% improvement over CUBIC. M2HO also outperforms the other baselines by 56.06% to 62.17%. Among all types of handovers, M2HO offers the most significant reduction in mmWave-sub-6GHz handovers, with an 82.11% time reduction. M2HO does better than the baselines by 78.66-82.75%. This is because the high throughput in mmWave and large bandwidth differences lead to many losses due to buffer overflows. M2HO is capable of suppressing these losses and avoiding slow starts.

6.2.3 Overhead To achieve its benefits, M2HO only incurs a small overhead as a user space application. We use ps to monitor the resource usage of M2HO process. It consumes an additional 5.0% CPU usage and 0.6% memory. As mentioned in §5.2, the overhead could be further reduced if we consider modifying the kernel as an option.

6.3 Effectiveness of Key Components

6.3.1 Handover Prediction We evaluate the precision, recall, and "earliness" of our handover prediction Algorithm 1. Precision is the percentage of correct predictions of handovers. Recall is the percentage of "occurred handovers" that are successfully predicted. Earliness refers to the time difference between when the UE predicts a handover and when the handover execution starts. As discussed in §4.2, it is important that this time is large to ensure a sufficient time for preparation. The evaluation results for recall and precision for different types of handovers are shown in Table 4. For all types of handovers, we achieve 94.8% precision and 87.8% recall, which are 53.8% and 48.8% higher than naively treating all measurement reports as handover indicators. Our recall is 23.9% higher than LTE-VR, which is designed for 4G LTE. M2HO captures the events that include mmWave cell measurements, such as the B1 event indicating a switch from a standalone LTE to mmWave cell. This ensures that M2HO predicts mmWave-related cell switches with higher recall.

On the other hand, although being much higher than the baselines, the recall for mmWave-sub-6GHz handovers is a modest 60%. This is because some handovers happen without measurement reports. We examine the link-layer logs of those handovers that are not captured by M2HO prediction. We find that there are continuous transmission failures before these handovers, causing the UE to spontaneously disconnect from the mmWave cell and fall back to sub-6GHz. Thus, M2HO's event-based algorithm cannot capture such behaviors.

We note two things. First, one might wonder why M2HO does not consider link failures also as handover indications. This is because, unlike in a normal handover, consecutive link failures will cause the connection to be immediately dropped. In our traces, 80.3% of such transmissions are dropped in the next 10 ms, leaving no time for rwnd adjustment. Second, even if such behaviors are not captured, M2HO falls back to the vanilla CUBIC, inducing no extra overhead.

We evaluate how early a prediction is made, by measuring "earliness," the time difference between the prediction and handover execution, as shown in Table 5. M2HO's prediction leaves 128.7-233.7 ms for other components to prepare for an upcoming handover. Meanwhile, earliness for naively taking a handover command as the indication is only 43.8-144.3 ms.

6.3.2 Mitigating Losses after Handover We evaluate how well M2HO reduces the number of loss events after handovers. The results are shown in Figure 13. Among all handovers, M2HO ensures 85.2% incur no loss, compared to 44.2%, 55.9%, 60.3%, and 54.1% in CUBIC, Westwood, C2TCP, and Verus. The baseline methods are not designed to handle buffering in handovers and thereby avoid losses. When M2HO fails to eliminate the losses, 87.9% of cases are handovers that are not captured by our prediction. When a handover is correctly predicted, in 97.9% of the cases, we do not experience loss.

6.3.3 Unnecessary Duplicate ACK Handling We next examine the effectiveness of M2HO in mitigating reordering after handovers. For simplicity, we name every three consecutive ACKs as a congestion event. Mitigating reordering requires high precision and recall: the congestion events that M2HO identifies and prevents should indeed be unnecessary (i.e., due to reordering), while M2HO must capture as many unnecessary congestion events as possible to prevent cwnd reduction. The results are shown in Table 3. M2HO achieves a 95.3% precision overall, with 89.5%, 95.8%, 72.5% and 98.0% for the four types of handovers, respectively. M2HO's effective buffer overflow prevention mitigates real packet loss, resulting in

high precision. We see that the precision for mmWave-sub-6GHz handovers, is smaller compared to the others. This is due to the lower recall rate in mmWave-sub-6GHz handover prediction. Without an early prediction, M2HO delays the preemptive mechanism, incurs packet loss, and thus reduces the overall precision of reorder handling. For the successfully predicted handovers, the precision increases to 99.3%.

M2HO achieves a 98.6% recall across all handovers, and 97.5%, 99.3%, 98.9%, and 100% for the four handover types. It shows that most of the reordering behavior can be effectively captured and suppressed with the rules discussed in §4.4.

6.3.4 Accuracy of Estimating Available Bandwidth Lastly, we evaluate how accurately M2HO estimates link capacity. Specifically, M2HO's estimate must have high accuracy, as it relies on it to control the send rate (by manipulating rwnd) in a sub-6GHz cell after a vertical handover from a mmWave cell. We compute the throughput over 1 s sliding windows, and compute the average TCP throughput over the UDP throughput, named the "throughput utilization rate." On average, TCP CUBIC only achieves a 68.4% average throughput utilization rate (118.53 Mbps). Meanwhile, M2HO achieves a 151.29 Mbps average throughput, and the utilization rate reaches 87.3%. This is a 27.6% improvement over TCP CUBIC.

7 Related Work

Improving throughput at 5G mmWave PHY. Considerable research has been conducted on fully exploiting the potential of 5G mmWave links. Many papers attempt to address the challenges arising due to the physical characteristics of mmWave, such as improving signal-to-interference-plus-noise ratio [25], mitigating mobile blockers [36], and enhancing reliability and throughput through multi-beamforming [37]. Our work studies deployed mmWave in 5G and its impact on TCP, which is complementary to these efforts.

Improving application performance in 5G. Some research efforts [49, 63] target application performance over highly dynamic 5G networks. They observe that TCP applications cannot make full use of bandwidth, and suggest specialized tuning for such applications. [33] measures handover characteristics (frequency, delay), and proposes a solution for video streaming application. On the other hand, M2HO reveals the root cause of such deficiencies by studying interactions between TCP and 5G handovers, and proposes device-centric, handover-aware solutions for CUBIC.

Reducing 5G handover disruption. To improve TCP performance during mobility, some prior efforts [30, 54] propose fine-tuning the handover control parameters to reduce handover disruption time, which may improve overall TCP performance. Our findings reveal that the packet losses caused by 5G handover have a significant and prolonged negative impact on performance beyond the disruption time.

TCP sending rate control efforts. Some works leverage PHY information to estimate the available bandwidth for TCP window control dynamically [43, 59]. These approaches however fail to resolve the adverse effects from 5G handovers, and/or pose impractical or high-overhead requirements such as knowledge of SINR to rate mapping. Other works rely on transport layer information only, to optimize TCP sending rates in dynamic cellular networks [17, 65]. However, they are handover agnostic and the link characteristic changes after handover leads to inaccurate estimation.

Window control and handover prediction. Some concepts similar to those used by M2HO have been partially explored, but are not applicable in our context. [31, 57] advertise receive window as zero during handovers to prevent packet loss and timeouts. However, unlike M2HO, they cannot predict mmWave handovers and setting the receive window to 0 wastes large mmWave bandwidth, while also disrupting applications. Several machine learning based handover prediction algorithms have been proposed [19, 33, 47, 49]; however, they either work on the network side or require modifying apps. Similar to M2HO, [55] uses a signaling-message for 4G handover prediction, but works poorly for 5G (§6.3).

8 Conclusion

We conduct extensive measurements to uncover critical deficiencies that hinder achieving high throughputs in 5G due to mobility induced handovers. To address these shortcomings, we design, implement, and evaluate M2HO, a novel device-centric solution to mitigate the negative effects from handovers. M2HO maintains various states of handovers and mitigates associated issues through state-dependent actions. Notably, it operates as a transparent middle layer on the device side only, requiring no modifications to the firmware, transport layer, or 5G protocols. Our research shows that simply exposing 5G mobility events to TCP, allows the latter to intelligently adapt its behavior to enhance throughput in mobile scenarios. In the future, we plan to extend our analysis to understand how handovers affect other TCP variants.

Acknowledgement

We would like to express our gratitude to the anonymous reviewers and shepherd for their invaluable feedback. This research was sponsored partly by the OUSD(R&E)/RT&L and was accomplished under Cooperative Agreement Number W911NF-20-2-0267. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the ARL and OUSD(R&E)/RT&L or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation herein.

References

[1] 2005. linux/include/net/tcp.h at. github.com/torvalds/linux/blob/master/include/net/tcp.h.

[2] 2006. TCP: make cubic the default · torvalds/linux@597811e. https://github.com/torvalds/linux/commit/597811ec167fa01c926a0957a91d9e39baa30e64.

[3] 2022. T-Mobile Tops 3 Gbps with World's First Standalone 5G Carrier Aggregation Achievement. https://www.t-mobile.com/news/network/t-mobile-tops-3-gbps-with-worlds-first-standalone-5g-carrier-aggregation-achievement. [Accessed January 2024].

[4] 2023. ACCUVER XCAL. https://www.accuver.com/sub/products/view.php?idx=6. [Accessed Februrary 2023].

[5] 2023. Faroese Telecom and Ericsson set European 5G mmWave downlink speed record. https://www.ericsson.com/en/press-releases/3/2023/faroese-telecom-and-ericsson-set-european-5g-mmwave-downlink-speed-record. [Accessed January 2024].

[6] 2023. Here are all the US cities with 5G coverage. https://www.androidauthority.com/5g-cities-us-1105898/. [Accessed January 2024].

[7] 2024. 5G Coverage Map | T-Mobile. https://wholesale.t-mobile.com/5g-coverage-map/. [Accessed January 2024].

[8] 2024. 5G Phones: 5G Ultra Wideband with a 5G Phone | Verizon. https://www.verizon.com/5g/phones/. [Accessed January 2024].

[9] 2024. AT&T Rolls Out 5G+ Across the U.S. https://about.att.com/pages/5g-plus.html. [Accessed January 2024].

[10] 2024. GSMA Thrive: China Sets Its Eye On Millimeter Wave 5G. https://www.spglobal.com/marketintelligence/en/news-insights/research/gsma-thrive-china-sets-its-eye-on-millimeter-wave-5g. [Accessed January 2024].

[11] 2024. Verizon Coverage Map: Nationwide 5G and 4G LTE Network Cell Phone Coverage | Verizon. https://www.verizon.com/coverage-map/. [Accessed January 2024].

[12] 3GPP. 2021. TS 38.300: NR and NG-RAN Overall description; (3GPP TS 38.300 version 16.4.0 Release 16).

[13] 3GPP. 2021. TS 38.331: Radio Resource Control (RRC); Protocol specification (3GPP TS 38.331 version 16.3.1 Release 16).

[14] 3GPP. 2022. TS 23.502: NR; Procedures for the 5G System (3GPP TS 23.502 version 15.2.0 Release 15).

[15] 3GPP. 2022. TS 37.340: NR; Multi-connectivity; Overall description; Stage-2 (3GPP TS 37.340 version 15.16.0 Release 15).

[16] 3GPP. 2024. TS 38.104: NR; Base Station (BS) radio transmission and reception (3GPP TS 38.104 version 17.14.0 Release 17).

[17] Soheil Abbasloo, Yang Xu, and H. Jonathan Chao. 2019. C2TCP: A Flexible Cellular TCP to Meet Stringent Delay Requirements. *IEEE Journal on Selected Areas in Communications* 37, 4 (2019), 918–932. https://doi.org/10.1109/JSAC.2019.2898758

[18] Ahmed M Al-samman, Marwan Hadri Azmi, and Tharek Abd Rahman. 2019. A survey of millimeter wave (mm-Wave) communications for 5G: Channel measurement below and above 6 GHz. In *Recent Trends in Data Science and Soft Computing: Proceedings of the 3rd International Conference of Reliable Information and Communication Technology (IRICT 2018)*. Springer, 451–463.

[19] Ahmed Alkhateeb, Iz Beltagy, and Sam Alex. 2018. Machine learning for reliable mmwave systems: Blockage prediction and proactive handoff. In *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. 1055–1059. https://doi.org/10.1109/GlobalSIP.2018.8646438

[20] Venkat Arun and Hari Balakrishnan. 2018. Copa: Practical {Delay-Based} congestion control for the internet. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 329–342.

[21] Arjun Balasingam, Manu Bansal, Rakesh Misra, Kanthi Nagaraj, Rahul Tandra, Sachin Katti, and Aaron Schulman. 2019. Detecting if lte is the bottleneck with bursttracker. In *The 25th Annual International Conference on Mobile Computing and Networking*. 1–15.

[22] Ethan Blanton, Dr. Vern Paxson, and Mark Allman. 2009. TCP Congestion Control. RFC 5681. https://doi.org/10.17487/RFC5681

[23] Lawrence S Brakmo, Sean W O'Malley, and Larry L Peterson. 1994. TCP Vegas: New techniques for congestion detection and avoidance. In *Proceedings of the conference on Communications architectures, protocols and applications*. 24–35.

[24] Kevin Brown and Suresh Singh. 1997. M-TCP: TCP for mobile cellular networks. *ACM SIGCOMM Computer Communication Review* 27, 5 (1997), 19–43.

[25] Sherif Adeshina Busari, Shahid Mumtaz, Saba Al-Rubaye, and Jonathan Rodriguez. 2018. 5G Millimeter-Wave Mobile Broadband: Performance and Challenges. *IEEE Communications Magazine* 56, 6 (2018), 137–143. https://doi.org/10.1109/MCOM.2018.1700878

[26] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time. *Queue* 14, 5 (2016), 20–53.

[27] Mohammed S Elbamby, Cristina Perfecto, Mehdi Bennis, and Klaus Doppler. 2018. Toward low-latency and ultra-reliable virtual reality. *IEEE network* 32, 2 (2018), 78–84.

[28] Filipa Fernandes, Christian Rom, Johannes Harrebek, Simon Svendsen, and Carles Navarro Manchón. 2022. Hand Blockage Impact on 5G mmWave Beam Management Performance. *IEEE Access* 10 (2022), 106033–106049. https://doi.org/10.1109/ACCESS.2022.3211525

[29] Sally Floyd, Tom Henderson, and Andrei Gurtov. 2004. *The NewReno modification to TCP's fast recovery algorithm*. Technical Report.

[30] Vigneswara Rao Gannapathy, Rosdiadee Nordin, Nor Fadzilah Abdullah, and Asma Abu-Samah. 2023. A Smart Handover Strategy for 5G mmWave Dual Connectivity Networks. *IEEE Access* 11 (2023), 134739–134759.

[31] T. Goff, J. Moronski, D.S. Phatak, and V. Gupta. 2000. Freeze-TCP: a true end-to-end TCP enhancement mechanism for mobile environments. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, Vol. 3. 1537–1545 vol.3. https://doi.org/10.1109/INFCOM.2000.832552

[32] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS operating systems review* 42, 5 (2008), 64–74.

[33] Ahmad Hassan, Arvind Narayanan, Anlan Zhang, Wei Ye, Ruiyang Zhu, Shuowei Jin, Jason Carpenter, Z. Morley Mao, Feng Qian, and Zhi-Li Zhang. 2022. Vivisecting mobility management in 5G cellular networks. In *Proceedings of the ACM SIGCOMM 2022 Conference*. Association for Computing Machinery, New York, NY, USA, 86–100. https://doi.org/10.1145/3544216.3544217

[34] Ananya Hazarika and Mehdi Rahmati. 2023. Towards an Evolved Immersive Experience: Exploring 5G- and Beyond-Enabled Ultra-Low-Latency Communications for Augmented and Virtual Reality. *Sensors* 23, 7 (2023). https://doi.org/10.3390/s23073682

[35] Van Jacobson, Robert Braden, and Dave Borman. 1992. RFC1323: TCP extensions for high performance.

[36] Ish Kumar Jain, Rajeev Kumar, and Shivendra S. Panwar. 2019. The Impact of Mobile Blockers on Millimeter Wave Cellular Systems. *IEEE Journal on Selected Areas in Communications* 37, 4 (2019), 854–868. https://doi.org/10.1109/JSAC.2019.2898756

[37] Ish Kumar Jain, Raghav Subbaraman, and Dinesh Bharadia. 2021. Two beams are better than one: towards reliable and high throughput mmWave links. In *Proceedings of the 2021 ACM SIGCOMM*. ACM, 488–502. https://doi.org/10.1145/3452296.3472924

[38] 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.

[39] Junhyeong Kim, You-Jun Choi, Gosan Noh, and Heesang Chung. 2023. On the Feasibility of Remote Driving Applications Over mmWave 5G Vehicular Communications: Implementation and Demonstration. *IEEE Transactions on Vehicular Technology* 72, 2 (2023), 2009–2023. https://doi.org/10.1109/TVT.2022.3210689

[40] Rajeev Kumar, Athanasios Koutsaftis, Fraida Fund, Gaurang Naik, Pei Liu, Yong Liu, and Shivendra Panwar. 2019. TCP BBR for ultra-low latency networking: challenges, analysis, and solutions. In *2019 IFIP Networking Conference (IFIP Networking)*. IEEE, 1–9.

[41] Yuanjie Li, Chunyi Peng, Zhehui Zhang, Zhaowei Tan, Haotian Deng, Jinghao Zhao, Qianru Li, Yunqi Guo, Kai Ling, Boyan Ding, et al. 2021. Experience: a five-year retrospective of MobileInsight. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*. 28–41.

[42] Hyoyoung Lim, Jinsung Lee, Jongyun Lee, Sandesh Dhawaskar Sathyanarayana, Junseon Kim, Anh Nguyen, Kwang Taik Kim, Youngbin Im, Mung Chiang, Dirk Grunwald, et al. 2023. An empirical study of 5G: Effect of edge on transport protocol and application performance. *IEEE Transactions on Mobile Computing* 23, 4 (2023), 3172–3186.

[43] Feng Lu, Hao Du, Ankur Jain, Geoffrey M. Voelker, Alex C. Snoeren, and Andreas Terzis. 2015. CQIC: Revisiting Cross-Layer Congestion Control for Cellular Networks. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications* (Santa Fe, New Mexico, USA) *(HotMobile '15)*. Association for Computing Machinery, 45–50. https://doi.org/10.1145/2699343.2699345

[44] Saverio Mascolo, Claudio Casetti, Mario Gerla, M. Y. Sanadidi, and Ren Wang. 2001. TCP westwood: Bandwidth estimation for enhanced transport over wireless links. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking* (Rome, Italy) *(MobiCom '01)*. ACM, 287–297. https://doi.org/10.1145/381677.381704

[45] Ayush Mishra, Xiangpeng Sun, Atishya Jain, Sameer Pande, Raj Joshi, and Ben Leong. 2019. The great internet TCP congestion control census. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 3, 3 (2019), 1–24.

[46] 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.

[47] Arvind Narayanan, Eman Ramadan, Rishabh Mehta, Xinyue Hu, Qingxu Liu, Rostand A. K. Fezeu, Udhaya Kumar Dayalan, Saurabh Verma, Peiqi Ji, Tao Li, Feng Qian, and Zhi-Li Zhang. 2020. Lumos5G: Mapping and Predicting Commercial mmWave 5G Throughput. In *Proceedings of the ACM Internet Measurement Conference* (Virtual Event) *(IMC '20)*. ACM, 176–193. https://doi.org/10.1145/3419394.3423629

[48] Arvind Narayanan, Muhammad Iqbal Rochman, Ahmad Hassan, Bariq S Firmansyah, Vanlin Sathya, Monisha Ghosh, Feng Qian, and Zhi-Li Zhang. 2022. A comparative measurement study of commercial 5G mmWave deployments. In *IEEE INFOCOM*. IEEE, 800–809.

[49] Arvind Narayanan, Xumiao Zhang, Ruiyang Zhu, Ahmad Hassan, Shuowei Jin, Xiao Zhu, Xiaoxuan Zhang, Denis Rybkin, Zhengxuan Yang, Zhuoqing Morley Mao, Feng Qian, and Zhi-Li Zhang. 2021. A variegated look at 5G in the wild: performance, power, and QoE implications. In *Proceedings of the 2021 ACM SIGCOMM*. ACM, 610–625. https://doi.org/10.1145/3452296.3472923

[50] Brien Posey. 2019. Explore the Cubic congestion control provider for Windows. (2019). https://bit.ly/2VfhxoA.

[51] I Rhee, L Xu, S Ha, A Zimmermann, L Eggert, and R Scheffenegger. 2018. RFC 8312: CUBIC for Fast Long-Distance Networks.

[52] Kei Sakaguchi, Ryuichi Fukatsu, Tao Yu, Eisuke Fukuda, Kim Mahler, Robert Heath, Takeo Fujii, Kazuaki Takahashi, Alexey Khoryaev, Satoshi Nagata, et al. 2021. Towards mmWave V2X in 5G and beyond to support automated driving. *IEICE Transactions on Communications* 104, 6 (2021), 587–603.

[53] Luca Schumann, Trinh Viet Doan, Tanya Shreedhar, Ricky Mok, and Vaibhav Bajpai. 2022. Impact of Evolving Protocols and COVID-19 on Internet Traffic Shares. arXiv:2201.00142 [cs.NI]

[54] GP Spoorthi and MB Akkamahadevi. 2019. Handover mechanism in 5G mmWave band. In *2019 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*. IEEE, 772–778.

[55] Zhaowei Tan, Yuanjie Li, Qianru Li, Zhehui Zhang, Zhehan Li, and Songwu Lu. 2018. Supporting mobile VR in LTE networks: How close are we? *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 2, 1 (2018), 1–31.

[56] Irina Tsareva, Trinh Viet Doan, and Vaibhav Bajpai. 2023. A Decade Long View of Internet Traffic Composition in Japan. In *2023 IFIP Networking Conference (IFIP Networking)*. 1–9. https://doi.org/10.23919/IFIPNetworking57963.2023.10186393

[57] Nen-Chung Wang, Ying-Yuan Wang, and Shih-Chien Chang. 2007. A Fast Adaptive Congestion Control Scheme for Improving TCP Performance During Soft Vertical Handoff. In *2007 IEEE Wireless Communications and Networking Conference*. 3641–3646. https://doi.org/10.1109/WCNC.2007.667

[58] Zhenyu Xiao, Lipeng Zhu, Yanming Liu, Pengfei Yi, Rui Zhang, Xiang-Gen Xia, and Robert Schober. 2022. A Survey on Millimeter-Wave Beamforming Enabled UAV Communications and Networking. *IEEE Communications Surveys & Tutorials* 24, 1 (2022), 557–610. https://doi.org/10.1109/COMST.2021.3124512

[59] Xiufeng Xie, Xinyu Zhang, and Shilin Zhu. 2017. Accelerating Mobile Web Loading Using Cellular Link Information *(MobiSys '17)*. ACM, 427–439. https://doi.org/10.1145/3081333.3081367

[60] Dongzhu Xu, Anfu Zhou, Xinyu Zhang, Guixian Wang, Xi Liu, Congkai An, Yiming Shi, Liang Liu, and Huadong Ma. 2020. Understanding operational 5G: A first measurement study on its coverage, performance and energy consumption. In *ACM SIGCOMM*. 479–494.

[61] Lisong Xu, Khaled Harfoush, and Injong Rhee. 2004. Binary increase congestion control (BIC) for fast long-distance networks. In *IEEE INFOCOM 2004*, Vol. 4. IEEE, 2514–2524.

[62] Tao Yu, Yoshitaka Takaku, Yohei Kaieda, and Kei Sakaguchi. 2021. Design and PoC Implementation of Mmwave-Based Offloading-Enabled UAV Surveillance System. *IEEE Open Journal of Vehicular Technology* 2 (2021), 436–447. https://doi.org/10.1109/OJVT.2021.3124787

[63] Xinjie Yuan, Mingzhou Wu, Zhi Wang, Yifei Zhu, Ming Ma, Junjian Guo, Zhi-Li Zhang, and Wenwu Zhu. 2022. Understanding 5g performance for real-world services: A content provider's perspective. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 101–113.

[64] Yasir Zaki, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. 2015. Adaptive Congestion Control for Unpredictable Cellular Networks. *SIGCOMM Comput. Commun. Rev.* 45, 4 (aug 2015), 509–522. https://doi.org/10.1145/2829988.2787498

[65] Yasir Zaki, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. 2015. Adaptive Congestion Control for Unpredictable Cellular Networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (London, United Kingdom) *(SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 509–522. https://doi.org/10.1145/2785956.2787498

[66] Menglei Zhang, Michele Polese, Marco Mezzavilla, Jing Zhu, Sundeep Rangan, Shivendra Panwar, and Michele Zorzi. 2019. Will TCP work in mmWave 5G cellular networks? *IEEE Communications Magazine* 57, 1 (2019), 65–71.