BISCAY: Practical Radio KPI Driven Congestion Control for Mobile Networks

Jon Larrea
Revelare Networks, USA

Tanya Shreedhar
Networked Systems group, TU Delft, Netherlands

Atte Niemi
Informatics, University of Edinburgh, UK

Adel Sefiane
Imperial College London & NVIDIA, UK

Mahesh K. Marina
Informatics, University of Edinburgh, UK

Abstract

Mobile application performance is often bottlenecked by cellular links with rapid bandwidth fluctuations. We show that radio KPIs from the device chipset can precisely and promptly measure available cellular bandwidth. Building on this, we propose Biscay, a practical KPI-driven congestion control for mobile networks. Biscay leverages OpenDiag, an in-kernel, real-time KPI extractor we introduce along with a KPI-based bandwidth estimator to adjust the congestion window, utilizing available bandwidth while minimizing delay. We implement Biscay and OpenDiag on unrooted Android 5G phones. Across trace-driven emulations and real-world 4G/5G experiments, Biscay outperforms state-of-the-art CCAs (e.g., BBR, CUBIC), typically reducing average and tail delay by >90% while matching or improving throughput. These gains stem from OpenDiag's 100× finer on-device KPI granularity than existing alternatives like MobileInsight.

2012 ACM Subject Classification Networks → Mobile networks; Networks → Transport protocols; Networks → Network resources allocation

Keywords and phrases Cellular Networks, Congestion Control, LTE/5G

Digital Object Identifier 10.4230/OASIcs.NINeS.2026.15

Acknowledgements I want to thank Aita.

1 Introduction

Mobile cellular networks enable ubiquitous, on-the-move connectivity for end devices. Global mobile subscriptions already exceed 8 billion, dwarfing fixed broadband, with networks rapidly shifting from 4G to 5G [39]. Despite this rollout, measurement studies show 5G still falls short of the high throughput and low delay required by many applications [54, 43, 53]. Beyond traditionally downstream-dominated traffic, next-generation applications generate substantial uplink demand [53]. Cloud gaming, AR/VR, video conferencing, backup services, and live HD streaming require significant real-time uplink capacity, making robust uplink performance essential for user experience.

Our focus is high-performance transport for mobile networks, specifically improving congestion control (CC). CC must maximize delivered throughput while minimizing delay; we capture this with the power metric (throughput/delay) and the classic rule "keep the pipe just full, but no fuller" via the path BDP [57]. Tracking this operating point in mobile settings is difficult because (1) the cellular link is typically the bottleneck [101, 60, 99, 105, 97], and (2) the available cellular bandwidth fluctuates rapidly due to wireless channel dynamics,

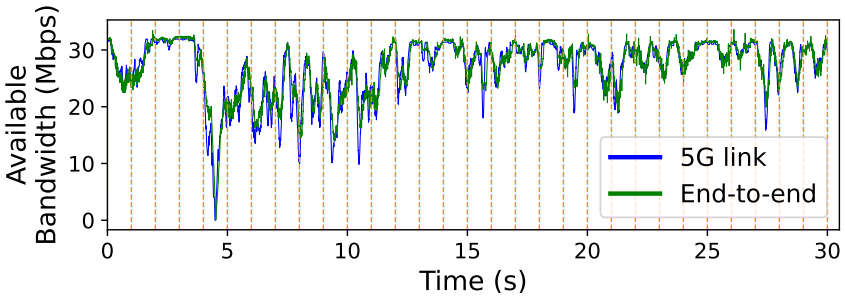


Figure 1 Measured trace of end-to-end and 5G link available bandwidth fluctuations while moving with a commodity 5G phone.

contention, mobility, and handovers [101, 60, 45, 105, 97]. Figure 1 illustrates both challenges using a trace from our dataset (commodity phone on a moving 5G connection; details in §6.3.1), highlighting the need for precise, timely knowledge of available bandwidth on the cellular segment.

Prior CC work for mobile networks (§3.1) falls into two broad categories: (i) estimation/prediction of available bandwidth (e.g., Sprout [97], PROTEUS [103], Verus [105], PropRate [62], ExLL [74]); and (ii) approaches requiring network support (e.g., ABC [45], DChannel [81]) or external sniffers (e.g., PBE-CC [101]). The former are inherently limited by estimation error, and the latter face deployability barriers.

Our key insight is that directly *measuring* available cellular bandwidth is better than estimating it, and routinely collected radio KPIs inside the device chipset can be leveraged for this measurement at no additional probing cost. Exploiting this raises four challenges: (1) selecting relevant KPIs and mapping them to available bandwidth; (2) extracting normally inaccessible KPIs efficiently and in real time; (3) integrating KPI-driven bandwidth into congestion window adaptation; and (4) achieving all of the above in a practical, deployable design.

We propose Biscay, a practical KPI-driven CC system that addresses these challenges. From 3GPP procedures [16], we identify KPIs (e.g., transport block size, number of physical resource blocks) that let the device determine available bandwidth on the cellular link. At the core of Biscay is OpenDiag, an in-kernel tool that uses the integrated Diag interface to extract arbitrary KPI sets in real time, enabling fine-grained bandwidth computation. Biscay then uses this cellular bandwidth to set the congestion window; if the bottleneck lies in the wired segment (less common), Biscay falls back to end-to-end bandwidth estimation. We implement Biscay and OpenDiag on Android devices and expose a user-space API via *libOD*. Unlike existing tools (e.g., MobileInsight [63, 64, 92]), OpenDiag provides two orders of magnitude finer KPI timescales without requiring root access. Combined with its device-centric design, Biscay is practical and readily deployable. Biscay-related code, tools, and scripts are available on Github [1].

We evaluate Biscay extensively against a wide range of CCAs using: (1) the Pantheon emulator [104] driven by numerous traces collected on commodity 5G phones augmented with OpenDiag (each trace includes backlogged UDP throughput and companion KPIs); and (2) real-world experiments on private and public 5G networks comparing Biscay to BBR [34] and CUBIC [48]. Table 1 summarizes key results (details in §6). Overall, Biscay significantly reduces average and tail delays (at least 50%, typically over 90%) while matching

[1] https://github.com/netsys-edinburgh/BISCAY

CCA	Tput	Avg Delay	Tail Delay	CCA	Tput	Avg Delay	Tail Delay
BBR	1.03×	58.51%	41.18%	PCC	1.84×	94.97%	95.01%
CUBIC	0.96×	98.74%	99.03%	Sprout	1.01×	80.69%	71.59%
Copa	1.01×	96.41%	96.48%	Verus	1.56×	90.83%	97.28%
LEDBAT	1.0×	92.9%	88.55%	Vivace	1.72×	93.03%	94.05%

Table 1 Summary of performance gains with Biscay (in terms of throughput increase factor and percentage of average/tail latency reduction) relative to existing CCAs.

or improving throughput. We also compare OpenDiag to MobileInsight [92] and the Android Telephony API [44], showing a 100× improvement in KPI granularity.

In summary, we make the following key contributions:

- **Biscay:** a device-centric cellular CC design that leverages radio KPIs from the mobile chipset (§4).
- **OpenDiag:** a real-time radio KPI extraction tool for commodity mobile devices that does not need rooting and allows arbitrary set of radio KPIs to be obtained from the radio modem at $10ms$ time granularity, that is an 100× improvement over common alternatives like MobileInsight [92] and Android Telephony API [44] (§4.2.3).
- **Evaluation:** in-kernel implementations on Android 5G phones (§5) and extensive emulation and real-world experiments demonstrating large delay reductions with comparable or better throughput (§6).

2 Background

This section provides a brief overview of the 5G networking stack on mobile devices and gives an overview of available on-device channels for communication with the radio modem.
Mobile Network Stack: Both 4G and 5G stacks are quite similar and reside under the IP layer in the TCP/IP model and provide similar functionality. For the sake of concreteness, however, we will focus on the 5G mobile network stack (illustrated in Figure 2).

Starting from the bottom, the Physical layer (PHY) [14] provides a transport channel to the upper layers and transfers higher layer information over the air interface to the 5G base station (gNB). Immediately above, the Medium Access Control layer (MAC) [11] serves as an interface between logical channels and the transport channel at PHY providing data transfer and radio resource allocation services to upper layers. The Radio Link Control layer (RLC) [17] sits on top of the MAC and is responsible for the transfer of upper layer Protocol Data Units (PDUs), error correction, concatenation, segmentation, reordering, duplicate detection and reassembly. Packet Data Convergence Protocol layer (PDCP) [13], the layer on top of RLC, is responsible for transferring user and control plane data, header compression, and ciphering/integrity protection. In between the PDCP and the IP layers, the Service Data Adaptation Protocol (SDAP) layer [19], a new addition relative to 4G, is in charge of the user plane traffic's quality of service. On the other hand, for the control plane, the Radio Resource Control layer (RRC) [18] configures the user and control planes according to the network state and is in control of the connection establishment/release, system information broadcast, radio bearer establishment/reconfiguration/release, mobility procedures (handovers) and paging notification. Finally, over the RRC layer, the Non-Access-Stratum layer (NAS) [12] is in charge of the session management procedures (authentication, security control, mobility, etc.) to establish and maintain IP connectivity between the device (UE) and AMF in the mobile core. The data communication between the device and remote endpoint happens via

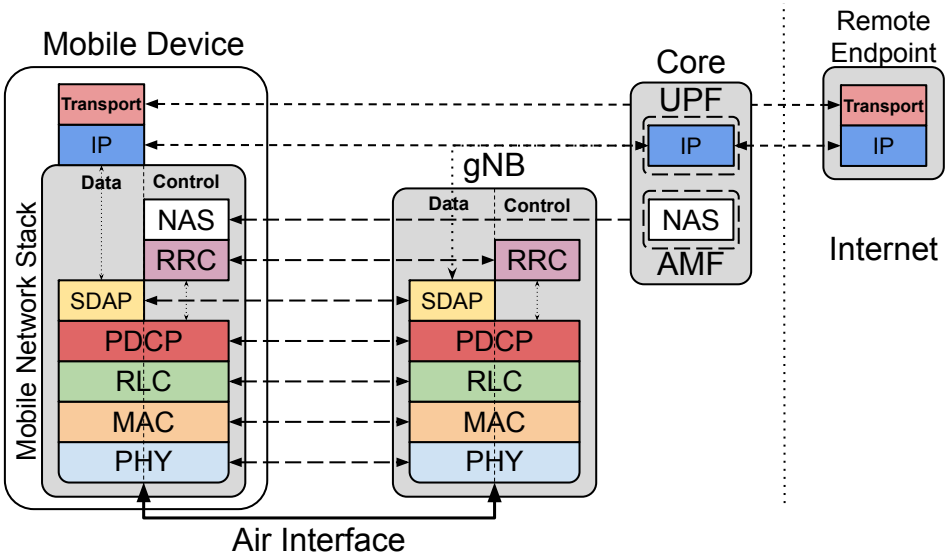


Figure 2 Schematic of 5G mobile network stack on device.

a tunnel to the UPF in the 5G core.

Communication with the Radio Modem: As pointed out in [63], the radio modem in a mobile device comes with a debug/diagnosis channel [67, 10, 76, 27, 38] that is primarily meant for Original Equipment Manufacturers (OEMs) to perform advanced baseband configurations and diagnostics. Here we focus on Qualcomm (Snapdragon series) modems given that they are the most common radio chipset in 5G devices. The diagnostic channel architecture in Qualcomm modems can be generalized to other manufacturers to a large extent. Every modern Qualcomm system on a chip (SOC) contains two different processing units: CPU and DSP. The CPU (typically an ARM based architecture) runs a general-purpose OS (GPOS) such as Android or iOS, whereas the DSP or modem (usually Hexagon based architecture [7]) runs a real-time OS (RTOS) such as Qualcomm QuRT RTOS. The GPOS and the RTOS are completely isolated from each other, and they can only interact with each other through a standard communication channel. This means that any process running in the GPOS (or even the GPOS kernel itself) cannot access anything within the RTOS or vice versa unless the standard communication channel is used. In Android, this channel is called Radio Interface Layer (RIL) [9], whose use is transparent to the user. The RIL defines a generic interface that applications (and even Android itself) use to interact with the modem. Some examples of the functions provided by the RIL are starting a call, terminating a call, introducing the SIM card pin and getting the coverage level. Given that RIL is a generic and modem agnostic interface, each modem manufacturer must provide a translation layer between the RIL and the specific modem that is referred to as the vendor-RIL, as illustrated in Figure 3. Qualcomm's vendor-RIL uses the standard RIL interface on one side and on the other side QMI (Qualcomm MSM Interface) [8] – a proprietary protocol used to interact with Qualcomm modems. From the perspective of radio KPI data collection, RIL offers only a small subset and that too a coarse time granularity (2-3s) via the Telephony API (a set of libraries built on top of RIL) [44].

Qualcomm modems additionally provide a side-channel called Diag (diag is also the name of the protocol) [72] for diagnostics and control. Unlike the RIL, diag was designed to provide all sorts of debug information and control capabilities so that manufacturers can use it to

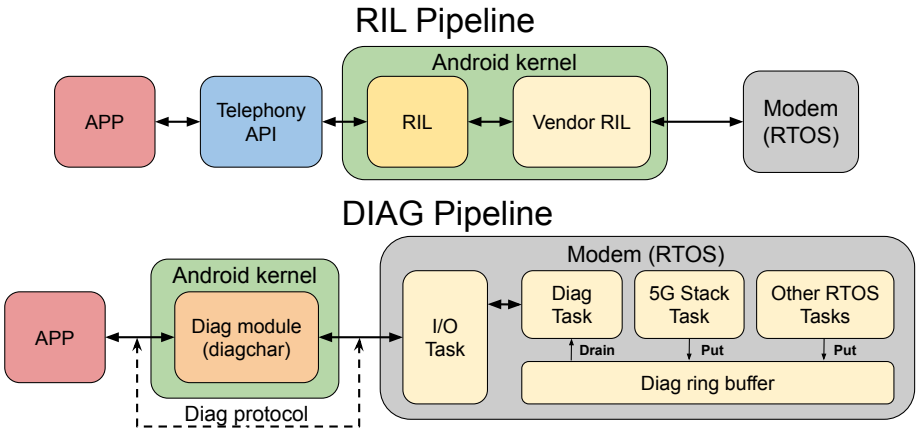


Figure 3 Architectures of RIL [9] (top) and DIAG [40] (bottom).

diagnose the modem using dedicated tools such as QXDM [77]. The only way of accessing the diag functionality is through the Diag kernel module (*diagchar*), an open-source kernel module provided by Qualcomm that acts as a shim between an application and the chipset, and exposes only the basic functionality (read, write and minimum protocol configuration). In practice, the Diag module is just a proxy that simplifies the access to the chipset. The bottom part of Figure 3 shows a schematic of the DIAG architecture. The application must implement the undocumented and proprietary diag protocol logic to communicate effectively with the chipset. Broadly speaking, the diag protocol offers two sets of features: gathering features (read and parse debug messages coming from the chip) and control features (modifying the chip's behavior and state). The former set of features have been partially reverse engineered and implemented by some KPI collection/measurement tools. The latter set of features have not previously been exploited by any measurement tool but enable an application to modify the behavior and state of the chip (change internal variables or disable the internal message buffering).

Carrier aggregation: Carrier aggregation (CA) [20] is a technique introduced in LTE-Advanced and remains an integral part of 5G for increasing the per-user bandwidth and the user throughput via aggregation of radio resources in the form of frequency blocks (called component carriers) from multiple cells and assigning them to the UE. CA is used when the amount of data to be transferred for the UE is insufficient with the resources from one cell, Primary Component Carrier (PCC), which is when the base station activates new cells or Secondary Component Carriers (SCC) to cope with that additional load. Though the SCCs are added and removed as needed, the PCC only changes at handover; the UE relies on the PCC for the RRC connection and to send/receive NAS information (e.g., security parameters). CA scenarios are common in dense urban environments where the number of available cells is higher.

3 Related Work

3.1 Congestion Control Mechanisms

End-to-end approaches. Loss-based CCAs (NewReno [47], CUBIC [48]) reduce `cwnd` on loss and react too slowly to fast cellular fluctuations, building queues and delay. Delay-based schemes (Vegas [31], FastTCP [95], Copa [29], LEDBAT [82]) operate at RTT timescales and miss sub-RTT air-interface variations, leading to inefficient utilization. Hybrid designs

(BBR [34], TCP-Illinois [65], Compound [89]) improve utilization but exhibit issues such as RTT unfairness, TCP unfriendliness, and robustness limits. Learning-based CCAs (PCP [26], PCC [36], PCC-Vivace [37], Remy [96], Indigo [104], Orca [23]) incur substantial training cost and generalize poorly across heterogeneous networks, complicating deployment.

Cellular-oriented approaches. Several works target cellular links explicitly (Sprout [97], PROTEUS [103], Verus [105], PropRate [62], ExLL [74], ABC [45], PBE-CC [101], DChannel [81]). Black-box estimators differ in how they track dynamics: Sprout/PROTEUS forecast short-term capacity; PropRate continuously probes; Verus uses a BBR-like delay signal tailored to cellular links; ExLL monitors packet arrival patterns at the receiver. mBBR [106] refines BBR for high-loss, rate-limited mobile links. LDRP [91] reduces uplink latency via application-layer dummy probing, duplicating transport-layer functions (e.g., BBR) and adding network/energy overheads. Other proposals assume infrastructure or hardware support such as CQIC [66], ABC [46, 45], XRC [55], and DChannel [81] rely on cross-layer base-station information (akin to AQM [70]), requiring changes at both BS and UE, hindering deployment. Sniffer-based designs (piStream [98], PBE-CC [101]) extract low-layer information using external SDRs [87, 3, 58, 33, 41, 100]; PBE-CC's reliance on non-3GPP functionality [21, 16, 15] and brute-force recovery makes it impractical for energy-constrained devices.

Compared to the existing solutions, 1) Biscay can adapt to the frequent and unpredictable fluctuations of the wireless bandwidth unlike end-to-end approaches which rely on RTT, resulting in fewer queue buildups and thus lower delays; 2) Biscay provides significantly more accurate bottleneck bandwidth estimate compared with the cellular-specific solutions that treat the air interface as a black box, resulting in throughput maximization while keeping the delays at a minimum; and 3) Biscay's device-centric design eliminates the need for support from the routers on the path or external devices (such as sniffers), facilitating its deployment and adoption.

3.2 On-Device Mobile Network Monitoring

Most commercial on-device mobile network monitoring tools function as trace-collector tools (e.g., [77, 24, 52, 51]) and are designed for offline analysis, where the data processing happens on a separate machine. Alternatively, commercial tools capable of *online monitoring* [75, 79, 85] are limited to on-device visualization of the data for service quality assessment or RF troubleshooting during field testing. However, they lack the ability to forward the collected data to another on-device consumer in real-time. In the open-source realm, some tools (e.g., [30, 71, 73]) dump in near-real-time basic information from some of the mobile layers for use on an external machine connected to the phone.

Among on-device KPI extractors, Telephony API [44] and the minimal driver prototypes of the Qualcomm Diag protocol [72] (e.g., [86, 42, 28, 68]) restrict KPI range and granularity (also shown in §6). MobileInsight [63, 64, 92] can access chipset KPIs but requires root (introducing vulnerabilities [90]). Furthermore, its online KPI extraction suffers from coarse (1s) data granularity due to its original offline design [4], necessitating user extensions to its mobile app [5] to be able to forward KPI data to other on-device consumers. It has been used in CLAW [99] and PERCEIVE [60] for specific applications, yet its coarse sampling limits control performance (§6). In contrast, OpenDiag provides fine-grained (10 ms) on-device, online radio KPI extraction without rooting, enabling effective cellular CC and other KPI-driven applications.

4 Design

We leverage ongoing KPI measurements from device radio chipsets to determine available cellular link bandwidth in a timely manner and improve cellular congestion control. We first outline the challenges, then present Biscay and its components, including the real-time KPI extractor OpenDiag.

4.1 Challenges

Cellular link bandwidth determination. The mobile device's 4G/5G chipsets capture many KPIs at millisecond granularity. Some like Channel Quality Indicator (CQI) are vital for mobile network and base station functions (e.g., MAC resource scheduling), while others aid in device-side monitoring and diagnostics (e.g., radio measurements for drive test minimization [22]). Our primary challenge is to identify the specific subset of radio KPI measurements from the device chipset that are important for calculating cellular link capacity and available bandwidth.

Real-time radio KPI extraction. In the context of this work, where available cellular link bandwidth is derived from low-level radio KPIs on devices, the age (time since measurement) and granularity (measurement frequency) of these KPIs significantly impact the precision of current bandwidth estimation. Fresh and finely-grained measurements are critical for accuracy. However, as discussed in §3.2, existing on-device radio KPI monitoring tools do not meet this requirement. Our second challenge is to solve this.

On-device radio KPI based congestion control. Apart from the above two challenges, we need to identify how and when to use the cellular link bandwidth measurement information from the congestion control perspective. Particularly when the cellular link is the bottleneck, the challenge is incorporating the measured cellular link bandwidth value for congestion control across multiple concurrently active flows.

Deployability. The ease of deployment of a congestion control (CC) design plays a key role in its widespread acceptance and adoption. Currently, the only CC design that leverages real-time radio KPIs requires bulky external hardware in the form of cellular sniffers [101] plugged into a phone, rendering it impractical for deployment. Ideally, both the CC system design and the radio KPI extraction framework should seamlessly operate on standard devices, without burdening users or necessitating device rooting.

4.2 Biscay

4.2.1 System Overview

Figure 4 overviews Biscay. OpenDiag, the *real-time KPI extraction layer*, interfaces with the Diag module to collect KPIs across the stack. Biscay's *cellular bandwidth determination layer* uses these KPIs to compute available bandwidth. Together with end-to-end bandwidth estimation, this feeds a *bottleneck determination layer*. The chosen bottleneck bandwidth is combined with RTT (from the Linux TCP stack) to set `cwnd` in the kernel. All layers except OpenDiag constitute the Biscay CC module.

4.2.2 Cellular link bandwidth determination layer

Unlike prior work that correlates/predicts bandwidth from raw KPIs [60, 99, 83, 84, 101], we replicate the modem's 3GPP procedures to compute available bandwidth, which is more accurate and robust (§6). We focus on 5G (4G is analogous).

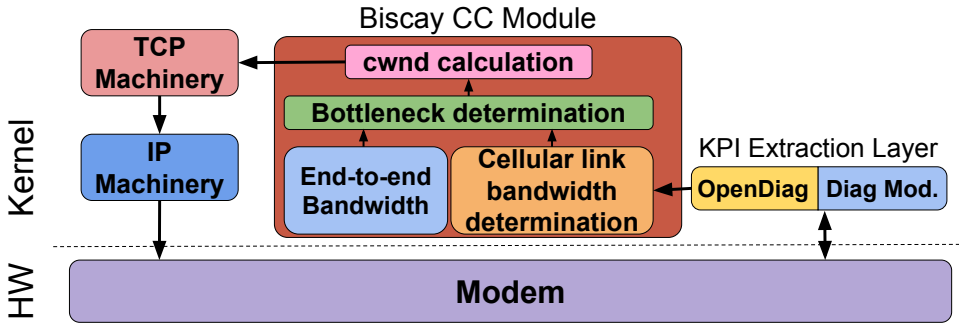


Figure 4 BISCAY congestion control system overview.

The 3GPP specification TS 38.306 - 4.1.2 [1] defines peak UE throughput from inputs such as carrier aggregation, MIMO layers, modulation, and resource block allocation. However, this formula does not accurately represent the current available bandwidth, it estimates achievable throughput under ideal conditions. 3GPP procedures [16] describe how UEs derive uplink bandwidth using Transport Block Size (TBS) granted via Downlink Control Information (DCI). In 5G, TBS index is computed per TTI (TS 38.214 [16]) from parameters such as MCS, redundancy version, and scheduled OFDM symbols. In 4G the TBS index is precomputed from MCS index. Given the TBS index and number of PRBs, throughput is obtained via pre-defined tables and multiplied by the number of antennas (MIMO). Carrier aggregation sums per-carrier values. In practice, DCI (via the diagnostic channel) already exposes TBS index and PRBs, eliding the first phase. We compute bandwidth per TTI:

$$bw = \sum_{c=1}^{Carriers} (tputTable[PRB(c), TBS(c)] * numAntennas) \quad (1)$$

Alternatively, the MAC layer reports a diagnostic summary of granted bytes and its utilization. This aggregates carriers/MIMO/standards (4G, 4G+, 5G NSA/SA) at a coarser 100 ms granularity. We evaluate both the granted-bytes KPI and the 3GPP-based calculation from a CC perspective in §6. DCI also carries downlink grants (TBS, PRBs). Using Eq. 1 with the 3GPP downlink tables [1] yields available downlink bandwidth.

4.2.3 OpenDiag: Real-time radio KPI extraction layer

Online bandwidth control requires KPIs immediately after chipset generation. Existing designs (MobileInsight [63] in Fig. 5) lack real-time capability due to three limitations:

- *Inter-process communication.* MobileInsight is a user-space application that is made up of two processes communicating via a pipe: diag_revealer (a C application responsible for the message collection) and MobileInsight App (a Java application with a Python interpreter on top responsible for the message processing). The data (diag messages containing KPIs) initially traverse the Diag module which interfaces to the modem diagnostic channel and acts as a data forwarder. In addition to these three entities that are part of the processing chain, MobileInsight must be extended in order to forward the processed KPIs to a consumer application (CC in our case), creating another step in the chain.
- *Processing time.* MobileInsight's packet processing framework parses and extracts a wide range of KPIs from packets. This can vary from a few tens of KPIs in small packets to several hundred or even a thousand KPIs in larger ones. However, for specific applications

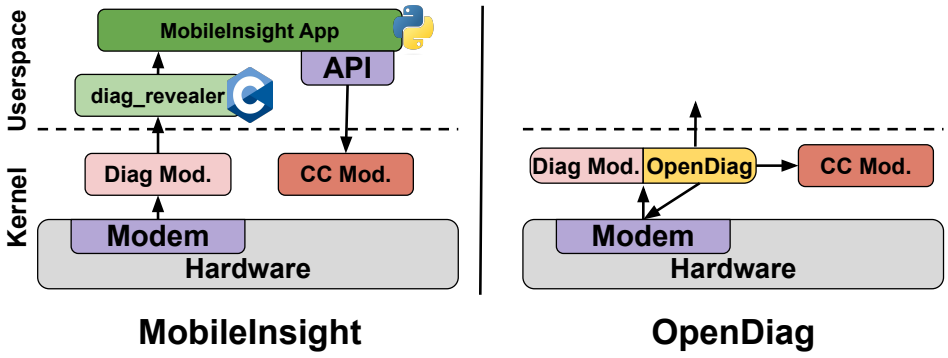


Figure 5 MobileInsight vs OpenDiag architecture.

like ours, this design becomes inefficient, as it necessitates waiting for the processing of all KPIs in selected packets before retrieving the required ones. This is particularly problematic when only a handful of KPIs are necessary.

- *Message granularity.* MobileInsight employs the Diag module to receive packets from the modem. However, this approach lacks insight into the modem's internal workings. Notably, it does not account for the modem's aggressive buffering mechanism, resulting in the release of message batches roughly every second, as demonstrated in §6.

OpenDiag enables real-time KPI retrieval (Fig. 5) via:

Tightly Coupled Architecture. In order to minimize the inter-process communication, OpenDiag tightly integrates with the Diag module (and its equivalent in modern Android versions). This design choice not only eliminates intermediaries like MobileInsight's diag_revealer but also grants access to private Diag module functionality (through exported symbols and bypassing non-essential features) that cannot be accessed as a user. This compact architecture design mandates that OpenDiag must run within the kernel. OpenDiag is shipped as a kernel module, so the kernel needs no modifications. This in-kernel design avoids the data from crossing the kernel-user boundary twice before reaching the CC module, generally deployed as a kernel module as well. Notably, eBPF-based strategies [2] prove ineffective in this context, the diagnosis logic resides in the modem and eBPF codelets cannot hook into external hardware.

Efficient Processing Framework. Unlike in MobileInsight, where to retrieve one KPI, all the KPIs within that packet have to be processed due to a packet-focused processing framework, OpenDiag employs a KPI-focused strategy in its processing framework. This approach stipulates that the smallest parsing unit is a single KPI (in contrast to a full packet), enabling the parsing of individual KPIs. The processing efficiency of our KPI-focused methodology is detailed in §6.

Establishing Control Channel. Due to its privileged access to Diag module's internal functionality (through the in-kernel architecture), OpenDiag is able to reach and manipulate modem internals inaccessible from user space. This establishes a control channel with the modem, serving the purpose of coordinating when to drain the internal buffer where the diagnosis messages get queued, effectively removing a key root cause of coarse granularity experienced by prior designs. Moreover, this control channel directs the chipset to offload the minimal necessary packet subset for extracting the user-defined KPIs.

4.2.4 Radio KPI based congestion control

Accurate bottleneck bandwidth estimation is central to CC. Biscay is a radio-KPI-based TCP CCA that determines cellular bandwidth in real time via OpenDiag and integrates with TCP (Fig. 4). OpenDiag supplies KPIs; the CC layer converts KPIs to throughput (§4.2.2) and, together with an RTT estimate, sets `cwnd`.

Bottleneck localization. Biscay continually maintains both an end-to-end bandwidth estimate (e.g., packet sending rate, ACK-based, or loss-based) and a wireless-link estimate via KPIs. When the cellular link is the bottleneck, the KPI-derived bandwidth is $\leq$ the end-to-end estimate; `cwnd` matches the cellular bandwidth, maximizing throughput while avoiding edge queue buildup. If the end-to-end estimate is lower, the bottleneck lies in the wired segment; Biscay falls back to any wired-specific CCA during that interval. The overhead of tracking two estimates and switching modes is negligible (few bytes of state and simple arithmetic).

Inter-flow fairness. Cellular base stations maintain per-UE deep buffers (no inter-user unfairness), but flows from the same UE share those buffers, creating an inter-flow fairness problem. Each flow runs its own CCA instance using only per-flow context. Biscay takes a device-level view: it tracks the number of active flows and apportions the KPI-derived bandwidth across them. The split policy is implementation-defined; standard scheduling algorithms from the literature can be used.

Downlink. Bandwidth determination for the cellular downlink follows §4.2.2. To relay this to the remote sender, we leverage TCP's built-in flow control (outside the congestion control layer) to throttle the sender to the cellular link bandwidth, avoiding queue buildup at the UE. Biscay modifies only the UE; using the relayed bandwidth to *increase* the sender's rate would require sender-side TCP changes. Prior downlink-focused proposals adopt this approach [101, 97]. Without a downlink bottleneck, ramp-up follows standard behavior.

5 Implementation

The implementation of Biscay has two core components: the *KPI extraction layer (OpenDiag)* and the *Biscay CC module*. We implemented both these components as kernel modules ($\approx$ 2500 lines of C code). We have developed multiple versions of OpenDiag for different Android versions. At the time of writing, OpenDiag has been tested and validated on Android 11, 12, 13, and 14. But given our extensive experience working with Android 11, our description and experiments are based on that version; it runs on top of the Linux Kernel 4.15 (default for Android 11 on Google Pixel 5). Besides, OpenDiag has also been validated on multiple Android devices and external modems such as Nexus 6P, Samsung Galaxy Note 4, OnePlus Nord 5G, OnePlus Nord N30, Nothing Phone 2, and Quectel modems. OpenDiag has also been used commercially for over a year to collect cellular data across 20 different countries spanning America, Europe and Asia. Additionally, for the sake of simplicity, we package these as part of a Custom Android update, facilitating deployment on non-rooted devices via a manual update. The method for manually updating the OS using a custom image varies across device vendors but in general, this can be done from the Android settings, through an App provided by the vendor or using Android's Fastboot mode.

KPI extraction layer. Biscay's KPI extraction layer is named OpenDiag and consists of a multi-threaded kernel module. The first kernel thread is responsible for obtaining and processing data received from the Diag module. This thread handles packet parsing and extracts any KPIs that the CC specifies. The second kernel thread is responsible for the control channel. It is configured to periodically instruct the modem to drain the internal

buffer where the diagnosis messages are queued. The control thread flushes the internal buffer every $1ms$ – this frequency was chosen to be smaller than the most frequent diagnostic message, which is generated every $10ms$.

Although Biscay will use OpenDiag from within the kernel, OpenDiag offers a subscription-based API for both kernel and user space consumers. This API is used by the CC layer (the consumer) to specify a list of KPIs that OpenDiag must extract and forward in real-time. The KPI forwarding is done through a shared memory channel in order to avoid excessive message passing within the kernel. This channel is realized through a shared buffer (memory-mapped to the user space application's virtual address space in case of accessing it from user space) that is allocated by OpenDiag based on the number of KPIs specified by the consumer. To avoid unnecessary busy waits (spinlocks), OpenDiag implements triple buffering over the shared-memory region so that the consumer can read KPI asynchronously without causing any race condition. To facilitate user-space interaction (such as user-space TCP or QUIC), we implement *libOD*. libOD exposes a simple POSIX-style API for applications to access radio KPI readings. libOD maintains a set of supported KPIs that can be easily extended to support obtaining any data from any diag message. At the time of writing, libOD supports the majority of relevant KPIs from PHY and MAC layers, the entire RRC and NAS layers (including ASN.1 and L3 Tabular decoding) and a subset of RLC and PDCP layers. Importantly, libOD API has been designed to be compatible with MobileInsight parsers, enabling them to be deployed on top of OpenDiag if needed even though the set of KPIs supported by OpenDiag is larger than what MobileInsight offers. Furthermore, the accuracy of the measurements performed with OpenDiag has been validated against state-of-the-art commercial tools such as Qualcomm's QXDM [77] and Keysight's Nemo Handy [52].

Biscay CC module. Biscay has been developed from scratch as an independent CC module only borrowing the pluggable features described in §4 (end-to-end bandwidth estimation and fallback mechanism) from an existing CCA in order to reduce development time. We leverage BBR [34] to implement Biscay's pluggable functionality. In particular, we leverage BBR's *end-to-end* bottleneck bandwidth estimation as part of bottleneck bandwidth estimation as well as vanilla BBR for the fallback mechanism (when the bottleneck switches to the wired segment, vanilla BBR will be used). Our decision is to use BBR's built-in end-to-end bandwidth estimation and fallback is based on the fact that both existing works [101, 102] and our own evaluations show the superior performance of BBR compared to other end-to-end approaches in the wired segment. Other recent works [35, 106] also acknowledge this and leverage BBR as the underlying base framework. These functionalities are taken from the BBR version integrated within the Linux kernel 4.15. A detailed comparison between Biscay's code and BBR's code can be found in §A.2 along with a description of Biscay's internals.

Our implementation complies with the Linux kernel's TCP machinery and gets loaded in the kernel as a CC callback which gets invoked when certain congestion events (e.g. on receiving an ACK, timeouts, duplicate ACKs, or Explicit Congestion Notification) trigger it. Biscay CC module interacts with OpenDiag using the latter's API and converts the KPIs into available bandwidth in Mbit/s. Subsequently, this bandwidth is converted into the kernel's native format (packets/μs) and then translated into a congestion window length by multiplying it by the estimated round-trip time (RTT). The RTT is obtained directly from the kernel's TCP machinery, which maintains a continuously updated RTT estimate based on per-ACK measurements. We leverage the existing BBR formulation to combine the kernel's bandwidth representation with the estimated RTT, thereby deriving the congestion

window (cwnd) used for transmission. In a downlink scenario, the receiver-advertised window is computed in the same way, in which the downlink KPI–based bandwidth is combined with the estimated RTT (also obtained from the TCP machinery) using the identical formulation. Biscay lists the active flows, identifying each flow by the 4-tuple corresponding with source IP, destination IP, source port, and destination port. In order to achieve active flow tracking, Biscay overwrites, using a transparent shim, some callbacks of the kernel's socket structure to get notified when any operation is performed in all system's sockets. Given the nature of TCP flows in Android [78] (over 80% of the TCP flows have a lifespan over 10 seconds), we have opted for a bandwidth distribution policy where every flow receive an equal share of the available cellular bandwidth targeting a fair distribution of the available resources. However, we acknowledge and discuss in §A.2 that some scenarios could be negatively affected by this policy and propose alternative scheduling policies that would perform better in such scenarios.

Despite using functionality from BBR, we further discuss how to leverage/integrate other CCAs features within Biscay in §A.2 inspired by how BBR v2/v3 leverages extra signals (ECN and packet loss) from other CCAs [93] to complement its congestion window calculation method.

In addition to this Android implementation, we have also implemented an offline version of Biscay, for evaluations with the Pantheon emulator [104]. Given that Pantheon can only be executed on a computer (we used Ubuntu 18) and OpenDiag cannot run there (there is no modem and Qualcomm drivers), we implemented a trace-based version of Biscay that takes a bandwidth trace generated from KPIs and replays it within Pantheon, effectively mimicking what would happen in the real world within the device.

6 Evaluation

We conduct a comprehensive evaluation of Biscay. First, we evaluate OpenDiag, the KPI extraction layer used in Biscay in isolation, looking at three key parameters: granularity, performance and battery consumption. Then, we evaluate the accuracy of two bandwidth determination methods proposed for Biscay, identifying the optimal granularity. Finally, we compare Biscay with 10 other CCAs under different mobility and workload scenarios. All our experiments are conducted in networks with 4G and 5G coverage.

6.1 Biscay's KPI extraction layer

In this section, we discuss the key performance features of OpenDiag, our KPI extraction layer that plays a crucial role in the bandwidth determination accuracy. All the OpenDiag related experiments shown in this section have been conducted over commercial cellular networks.

Granularity. Granularity, is the frequency at which packets are received from the chipset, is a key feature that enables real-time data collection. Having a fine granularity means that you can extract samples of a given KPI at any moment in time more accurately. Figure 6a shows a comparison between six different collection tools (Nemo Handy [52], Telephony API [44], MobileInsight [63], QualiPoc [79], R&S TSMA6 [80] and OpenDiag) and the finest granularity at which the chipset can report a given KPI which is the ground truth. For this experiment, we chose Reference Signal Received Power (RSRP) as the extracted KPI. RSRP is a standard KPI used in multiple works, and it is one of the KPIs present in the most frequent packet generated by the chipset through the diag interface (LTE Serving Cell Measurement Response packet).

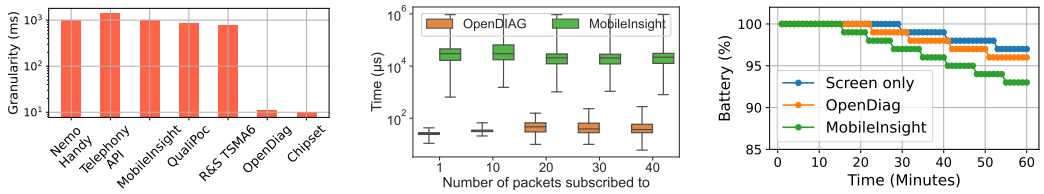


Figure 6 a) Granularity of different tools vs chipset (ground truth). **b)** Packet processing time of OpenDiag vs MobileInsight as a function of subscribed packets. **c)** Battery cost over time on COTS UE for Screen, OpenDiag and MobileInsight; both recording the required KPIs.

In Figure 6a, granularity is reported as the time between samples in ms. While the minimum packet granularity offered by commercial [52, 79, 80], standard [44] and open source solutions [63] is in the order of $1000ms$, OpenDiag is able to retrieve packets from the chipset almost every $10.9ms$, a $100\times$ improvement over all the alternative tools. Despite it's in-kernel design, the main reason behind such an improvement is the use of the control channel of OpenDiag that forces the chipset to drain the packets of the modem's internal buffer every ms. A similar improvement ($95\times$) has been obtained using a user-space version of OpenDiag, which rules out the kernel factor as the main reason for the improvement. The overhead created by the message gathering and parsing processes generates a 9% overhead over the ground truth with $10ms$ granularity. Interestingly, if the logs generated by MobileInsight are analyzed, the reported granularity matches the ground truth. However, this alignment is deceiving due to MobileInsight's time mechanism which uses the timestamp that comes in the header of each packet (the time at which the chipset created that packet) as the packet timestamp rather than the time at which the packet is received by the application. In practice, if MobileInsight is used and the time at which the application receives a packet is recorded, we will observe that a batch of packets is received at a given time due to the chipset's internal buffering. Roughly one second later, another batch of packets is received by the application, comprising those generated during that particular second. Extending this behavior to the value of a given KPI, we will observe that the value of the KPI remains unchanged for a second and, when the batch of packets is received and parsed, its value will change several times within a period of a few nanoseconds. In practice, this is seen as one-second granularity.

Processing time. Besides the granularity, the other key factor that prevents us from using MobileInsight as the KPI extraction layer is the processing performance rooted in its multi-layer design and inefficient processing pipeline.

In Figure 6b, we measure how long each packet spends in the processing pipeline of OpenDiag and MobileInsight. This is the time between the packet's arrival in the processing pipeline and the time it has been dispatched and is available for the consumer. Even though Biscay only uses a few KPIs, we conducted a comprehensive evaluation where we extract one KPI for every packet that we are subscribed to while we increase the number of packets. Please note that each packet type might contain multiple KPIs. Due to the limitations of MobileInsight's data forwarding to a third-party application, we implemented a dummy consumer within itself (OpenDiag's consumer was another application, incurring extra time to forward the KPIs). From the numbers shown in Figure 6b, it is clear that OpenDiag's design plays a significant role in terms of performance, i.e., timely KPI retrieval with improvements in the order of $100-1000\times$. While OpenDiag's additional delay remains in the order of tens of μs, MobileInsight's pipeline generates processing delays of hundreds of ms, making it completely unusable to calculate the grants received in real-time every TTI

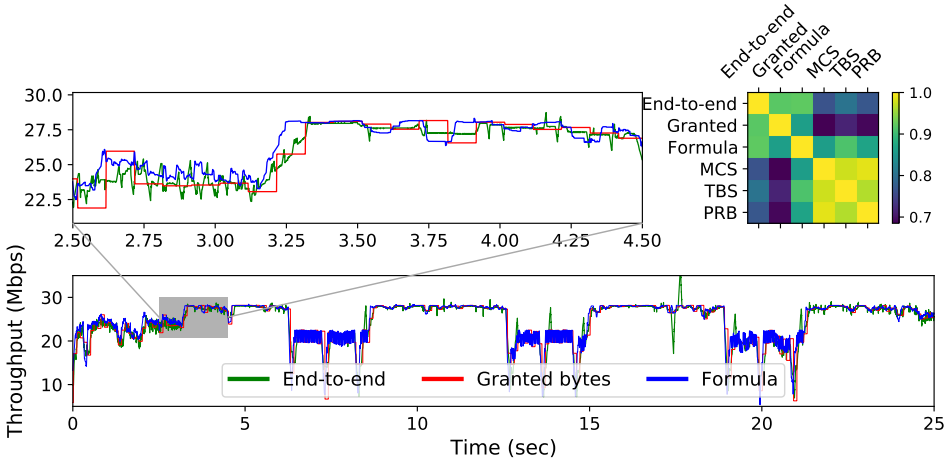


Figure 7 Throughput comparison of granted KPIs and 3GPP formula with end-to-end ground truth. Heatmap indicates the high Pearson correlation in the bandwidth determination methods.

(5G TTI is $0.5ms$).

Energy efficiency. BISCAY requires a KPI extraction layer to run in the background in order to calculate the bandwidth. Due to this, we conduct an evaluation of the battery life penalty that the user must pay for using OPENDIAG as a KPI extraction tool compared to MobileInsight. We discover that the modem only generates debug packets when it is in active mode. During the low-power mode, it generates very infrequent and periodic reports. So, to keep the UE in active mode, we are generating a small but constant data plane traffic using `iperf3`. Besides, for all our tests, we disabled Android's adaptive battery and screen brightness options to maintain consistent behavior between measurements.

Figure 6c shows the battery consumption over an hour of three different configurations: Screen only (the screen remains on while `ping` was running in the background), OPENDIAG (the screen was on and the required KPIs for BISCAY were recorded using OPENDIAG), and MobileInsight (the required KPIs were recorded using MobileInsigth). For all three configurations, the modem was in active mode. The results show that the battery degradation of OPENDIAG compared with the baseline (Screen only) is negligible over the 60-minute period. However, with MobileInsight, the battery consumption is more noticeable. We mainly attribute this to the multi-layer design used by MobileInsight, which requires multiple applications to run concurrently.

6.2 Biscay's bandwidth determination

The design section (§4) introduces two different ways of determining the maximum available bandwidth in the radio link: the simplified 3GPP throughput calculation and MAC layer granted bytes. Theoretically, the main advantage of the former method comes with its granularity (throughput can be determined at TTI granularity) with the trade-off of extracting multiple KPIs such as PRBs, TBS index, and MIMO for all the serving cells if CA is enabled. On the other hand, the advantage of using granted bytes comes from its simplicity (one single KPI contains the resulting throughput after considering 4G/5G, CA, MIMO, etc.); however, this KPI gets updated every $100ms$.

Figure 7 shows a correlation study between the two throughput determination methods defined by BISCAY with the ground-truth throughput that corresponds with the throughput received at the receiver side (server). We also include the raw KPIs used in similar works [60,

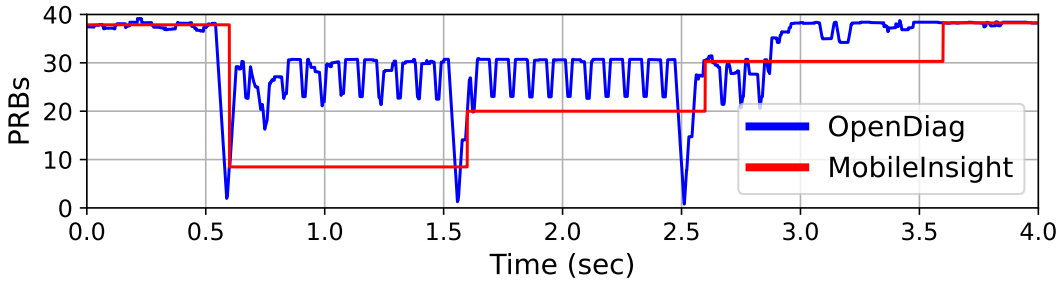


Figure 8 Measurement of PRBs with OpenDiag & MobileInsight.

99, 101] to estimate the bottleneck bandwidth (Biscay uses those KPIs as indexes in the pre-calculated 3GPP-defined tables). The correlation matrix clearly shows an extremely high Pearson correlation (over 0.95) with the bottleneck bandwidth of the two methods defined by Biscay. Interestingly, even though it suffers from a $100ms$ granularity, the granted bytes KPI can perform as well as the simplified 3GPP formula, suggesting that having extremely fine granularity (TTI-granularity) might not be a decisive factor in CC. This experiment also proves how simply using raw KPIs is not an accurate method to determine the radio link throughput, with the correlation coefficient barely reaching 0.75. A time-series plot of a randomly picked scenario (mobility on-peak with CA enabled and 4G+5G) complements the correlation matrix showing how similar to the ground truth both throughput determination methods are. Please note the propagation delay has been removed for visualization purposes.

Figure 8 shows the time series of PRB allocations using both OpenDiag and MobileInsight. Note that PRB is a common KPI used in similar works [99, 60, 101]. Due to the differences in update frequencies between these, we observe a clear distinction in the measured KPI (PRB) time series. The less frequent updates with MobileInsight causes applications relying on its measurements (e.g., PERCEIVE [60] or Claw [99]) to rely on outdated or less precise estimates during intervals between the updates. In contrast, the fine granularity provided by OpenDiag leads to accurate measurements, enabling more effective and optimal utilization of resources. The discrepancy between reported and actual real-time KPI data can significantly impact the performance and efficacy of applications reliant on KPI based estimates.

The combination of the proposed bandwidth determination method's accuracy (Figure 7) and precise KPI measurements resulting from a finer granularity sampling (Figure 8) hints towards optimal performance from transport layer point of view. Given this, we decided to conduct an experiment to identify the effects of the data granularity on the transport layer performance metrics (throughput, average and tail delay as of 95th percentile delay). The results depicted in Figure 9, show the variation of end-to-end transport layer metrics with the KPI sampling interval. We modified Biscay to sample the air interface at a given pre-defined frequency varied between $1 - 1500ms$. The scenario shown in Figure 9 is the same scenario shown in Figures 7 and 8. We added the results obtained by BBR as dashed horizontal lines as reference.

The results show that while the throughput doesn't decrease significantly as we increase the sampling interval, both average and tail delays do increase with the KPI sampling interval. This behavior implies that coarser sampling rates translate to Biscay saturating the channel, leading to queue buildups resulting in the same throughput and more delays. Interestingly, this behavior is only noticeable for sampling rates larger than $100ms$, which explains why there is no difference between the throughput determination methods evaluated in the correlation analysis. For sampling rates higher than $100ms$, Biscay starts to perform

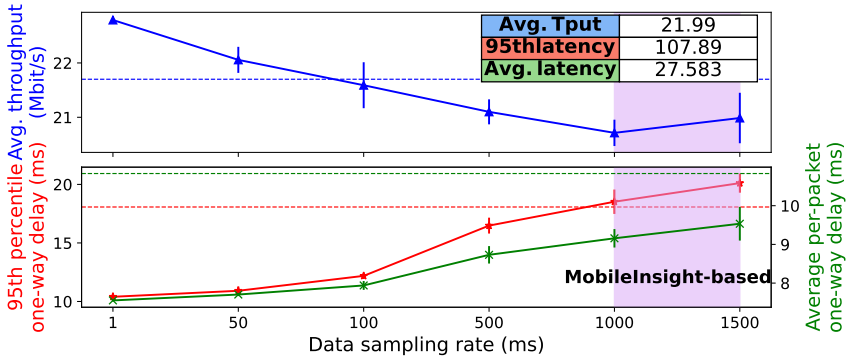


Figure 9 Effects of data sampling frequency on different transport metrics. The table on top shows BISCAY's performance using the average channel bandwidth.

like BBR and even underperforms BBR in both throughput and delay if the granularity is coarse enough. Both, throughput and delay performance get degraded even more when granularity beyond $1000ms$ is used (where all MobileInsight-based solutions [60, 99] operate). We have added a table in Figure 9 (at top) that shows BISCAY's performance using the coarser sampling rate, sampling the channel just once.

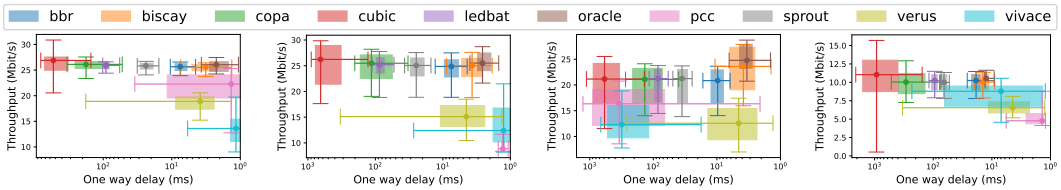
If this experiment is repeated multiple times, the result will be the same as using the average channel throughput in BISCAY. Consistent with the graph, the throughput remains within a constant range while the tail delay and average delay grow drastically to $107ms$ and $27ms$, respectively. These results and the required granularities highlight the need for a KPI extraction layer like OPENDIAG given that the best granularity that can be obtained with all the alternative solutions is $1000ms$. It also validates that there is no significant difference in transport performance between the two throughput determination methods besides the implementation complexity. Therefore, our presented implementation uses the MAC layer granted bytes KPI as a proxy to calculate the radio link bandwidth.

6.3 Congestion Control

We explore the performance of BISCAY in terms of throughput and delay (average and tail delays) in commercial 4G and 5G networks. We compare BISCAY with 8 other state-of-the-art CCAs. We deliberately picked a representative of each of the categories discussed in the related work for the comparison: CUBIC [48] (loss-based), Copa [29] & LEDBAT [82] (delay-based), BBR [34] (hybrid), PCC [36] & Vivace [37] (learning-based), and Sprout [97] & Verus [105] (wireless-aware). As an additional baseline, we include the Oracle, a CCA that precisely knows the end-to-end bottleneck bandwidth at any given point in time and so achieves the optimal performance (maximum throughput and minimum delay).

6.3.1 Real World Dataset Collection

To record the traces, we measured the uplink and downlink throughput across time by saturating the link (both directions individually) with MTU-sized UDP packets. This is because TCP cannot reliably saturate the channel as CC will kick in and reduce the sending rate. This methodology is consistent with prior research [81, 97, 94]. We recorded the throughput within the UE (Google Pixel 5, OnePlus Nord N30 and OnePlus Nord 10T) and in the receiver server using `tcpdump` to ensure the correctness of our measurements. `iperf3` was used within sender as a traffic generator. To extract the required KPIs, OPENDIAG was running in parallel with `tcpdump` in the UE. Those KPIs are later introduced in a trace



a) Mobility Off-Peak **b)** Mobility On-Peak **c)** Static Off-Peak **d)** Static On-Peak

Figure 10 Single-flow throughput and one-way delay (axis reversed) comparison of different CCAs in commercial networks. The boxes' horizontal edges (one-way delay) represent the 25th and 75th percentiles, and the ends of the error bar give the 10th and 90th (tail delay). Similarly, vertical box edges represent the 25th and 75th throughput percentiles, and the error bars give the 10th and 90th percentiles. The error bar intersection shows the throughput and delay medians. The top-right is the best performing throughput-delay pair.

that is fed into Pantheon §6.3.2. This setup was used to collect multiple traces under four different scenarios: mobility on-peak, mobility off-peak, static on-peak and static off-peak. The on-peak label corresponds to a trace collected during busy hours (9 am to 6 pm), whereas the off-peak label corresponds to traces collected between 10 pm and 2 am. We define a scenario as mobile when the UE is moving between cells (labeled as mobility traces recorded while walking, driving, bus, and train), and static corresponds to when the UE is not moving between cells. All data collection occurred within an urban or campus area in major cities in EU and US.

For a robust evaluation, we intentionally designed our data collection approach to transition between 4G and 5G (NSA and SA) covered areas during each measurement. So, every mobility trace includes both 4G and 5G data. Additionally, the mobility traces were recorded with CA enabled, reflecting the effect of being served by multiple cells. We collected multiple traces in different locations for the static case to reflect the diversity in network conditions (4G/5G and single/multiple serving cells). Our methodology aimed to provide a well-rounded view, ensuring the reliability and generality.

More information about the dataset, its characteristic and scenarios distribution can be found in A.1

6.3.2 Testbed Configuration

The evaluation is primarily conducted on the Pantheon emulator [104], a network emulation tool replaying pre-recorded network traces under emulated network conditions. Pantheon is built on top of mahimahi [69], another emulation framework initially designed for HTTP-based traffic. Both of these tools are widely used in the networking research community [97, 104, 101, 29, 103]. Pantheon is deployed in the Powder platform [32, 6] using a 32-core CPU machine with 64 GB of memory running Ubuntu 18. We generated 260 distinct Pantheon traces from our measurement campaign. The reason behind using Pantheon for a subset of the experiments is to ensure a fair evaluation where all CCAs are evaluated under the exact same conditions since that cannot be guaranteed in the wild.

6.3.3 Single-flow performance

Figure 10 compares Biscay with the eight CCAs mentioned above under four scenarios: Mobility On-Peak, Mobility Off-Peak, Static On-Peak and Static Off-Peak. The x-axis of the plot is reversed and the top right region is the best performing throughput-delay pair. Each graph is the average from the respective set of traces. It is important to highlight

		Two flows			Three flows		
CCA	Scenario	Avg Tput (Mbit/s)	Avg Delay (ms)	Tail Delay (ms)	Avg Tput (Mbit/s)	Avg Delay (ms)	Tail Delay (ms)
Biscay	Mobile	12.76	5.625	7.64	8.5	5.78	7.54
	Static	8.89	10.92	13.62	5.91	10.04	12.79
BBR	Mobile	12.59	7.68	9.73	8.42	7.91	10.05
	Static	8.46	14.35	19.69	5.54	17.01	25.15
CUBIC	Mobile	13.54	417.2	802.9	9.17	397.33	830.38
	Static	9.13	768.9	1536	5.91	667.34	1450.80

Table 2 Performance comparison with two and three simultaneous flows using Biscay, BBR and CUBIC.

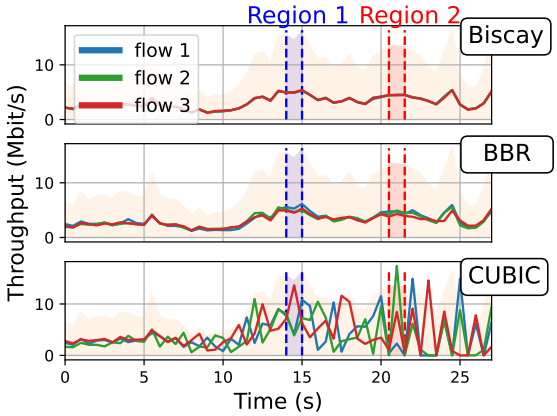
that Pantheon will determine the emulator delay based on the provided trace, which is not representative of the delay experienced during the measurement campaign. However, the queue delay ratio amongst the different CCAs is trustworthy.

Across all the scenarios, CUBIC has the highest delays (average and tail) because its cwnd gets reduced only when a loss is detected. Until then, the bottleneck queue builds up, leading to increased delays before the packets are dropped. Copa and LEDBAT, both delay-based, report throughputs similar to CUBIC and a fraction of CUBIC's delays across all the scenarios.

Sprout and Verus can significantly reduce the delay, given that they were specifically designed for wireless access networks. Interestingly, Sprout matches CUBIC's throughput, but Verus is far from that, suggesting that the source of its low delay (although extremely high tail delay) is due to not fully saturating the channel. Learning-based CCAs (PCC and Vivace) show abnormal and inconsistent behavior across the different scenarios for both throughput and delay, hinting that the models employed are overfitting in some scenarios. BBR results outperform all the previously discussed CCAs in all the scenarios, both in terms of delay (excluding overfits and abnormal results) and throughput. Finally, Biscay is able to maximize the channel usage, resulting in maximum throughput without paying delay penalties attributed to its accurate bottleneck bandwidth determination method that calculates the precise bandwidth in real-time and adjusts the sending rate accordingly. Moreover, among all the CCAs evaluated, Biscay is the closest to the Oracle, both in terms of throughput and delay in all the scenarios. The reason for this lies in the fact that the different bandwidth calculations used by the evaluated CCAs are inaccurate and coarse-grain approximations of the bottleneck bandwidth. Finally, an interesting observation is that, on average, all the evaluated CCAs seem to have better performance (more throughput and equal or lower latency) in mobile scenarios as opposed to static, which is counterintuitive. That is just an artifact of the dataset used; more on this in A.1

6.3.4 Multi-flow performance

For a comprehensive analysis of Biscay's performance, we conduct an experiment where multiple flows are simultaneously active. From the experiments shown in Figure 10 and summarized in Table 1, it can be seen that BBR is closest in performance to Biscay across all the scenarios. Additionally, CUBIC is the default CCA in the Linux operating system and, therefore, in Android. We consider these two CCAs for the multi-flow evaluation. The performance of the CCAs was evaluated under multi-flow conditions by simultaneously running two and three flows (Table 2). The idea behind this evaluation is that an end device



CCA	μ	μ_{reg1}	μ_{reg2}
Biscay	1.0	1.0	1.0
BBR	0.99	0.97	0.95
CUBIC	0.99	0.70	0.86

Figure 11 Per-flow throughput for Biscay, BBR and CUBIC. The shaded region indicates the total channel bandwidth. **Table 3:** Jain's fairness index of the different shadowed areas.

hardly runs a single flow and is configured with a given CCA, which gets applied at the OS level. Therefore all the TCP sockets open in that system will use the predefined CCA unless otherwise specified through the socket options, a practice rarely seen outside networking laboratories. For simplicity's sake, both tables contain the average results of On-Peak and Off-Peak for the Mobile and Static scenarios.

We observe similar behavior to that of single-flow experiments in terms of throughput. All three CCAs saturate the channel, and there is not much throughput difference between them. However, not all of them maximize channel usage at the same cost. CUBIC saturates the channel, which is reflected in high average and tail delays (up to 1.5s). BBR's more precise bottleneck bandwidth estimation leads to smaller queues at the bottleneck link, resulting in lower average and tail delays. However, BBR design does not consider other in-device flows when determining the cwnd of a given flow, which is reflected when flows compete for bandwidth, resulting in queue buildups and higher delays (average and tail) increases. BBR is more conservative and decreases the window whenever it detects queues being built. On the other hand, Biscay is aware of the number of active flows in the system, and it is able to divide the `grant received` by the base station, which is meant for the entire device evenly among the active flows avoiding the competition and limiting the queue buildup. Besides the apparent decrease in throughput, the flow-aware design of Biscay enables it to maintain similar or even lower average and tail delays. This demonstrates the efficacy of Biscay's design in managing multiple flows.

6.3.5 Fairness

Building on the multi-flow experiments where flows competed for bandwidth, we evaluated how fair Biscay is under such competition and compared it with BBR and CUBIC. Unlike WiFi networks, where devices contend for access to the same physical resources, in mobile networks, the base station manages the allocation of resources to the users. The base station's MAC scheduler makes resource allocation decisions based on the Scheduling Request received from the UE and the observed channel quality. The scheduler then grants a portion of the available resources to the UE through a Downlink Control Information (DCI) message. In commercial RANs, scheduling algorithms typically use a proportional fair approach to ensure fairness among users, as fairness is a crucial requirement in the RAN [56]. Additionally, unlike WiFi access points, where all users' data is queued in the same buffer, base stations have dedicated deep buffers for each UE. Therefore, inter-user fairness relies on the base

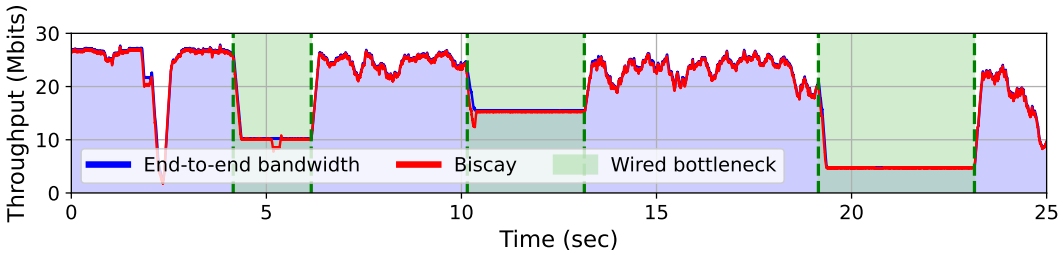


Figure 12 BISCAY falling back to BBR when the bottleneck changes to the wired segment

station scheduler, not the UEs. However, an inter-flow competition where multiple flows compete for the uplink bandwidth within the UE is still an issue that CCAs have to deal with.

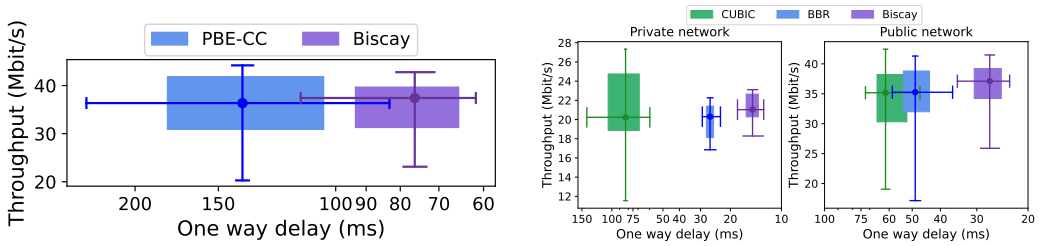
Figure 11 shows the throughput achieved by three simultaneous flows when running BISCAY, BBR and CUBIC as time series. The shaded area in the background represents the total channel bandwidth. In CUBIC, each flow strives to maximize its throughput, leading to a suboptimal allocation of resources. In contrast, BBR is more conservative, focusing on reducing delay, resulting in a more uniform distribution of the channel resources among active flows compared to CUBIC. Finally, BISCAY is able to split the available bandwidth equally among the existing flows, which is attributed to having a global view of the bandwidth available for the UE, ensuring that every flow gets the same amount of bandwidth from the total available. Additionally, we have calculated the fairness using Jain's fairness index [50]. Figure 11 (right) includes the fairness index of the entire experiment and the two shaded areas (Region 1 and Region 2). One of the limitations, as can be seen here, is that Jain's index uses the average throughput over the selected time period masking and hiding fine-grain details. This is proven by the fact that the fairness index of the entire experiment is near perfect for the three CCAs with an index equal to 1, while the time series show contradictory behavior. The fairness indexes of the two shaded regions show that even though the three CCA might look fair over long periods, BBR and CUBIC bandwidth distribution is unfair (particularly harmful in short-duration communications such as web traffic).

6.3.6 Bottleneck detection

Finally, we assess BISCAY's ability to detect changes in bottleneck location and its intended behavior of falling back to a wired-specific CCA (our implementation uses BBR) when the bottleneck shifts to the wired segment of the path. Although we did not encounter any instances of wired-segment bottlenecks in our measurements and experiments on public networks, we intentionally simulated this scenario. This involved manually limiting the bandwidth of the wired segment to a value lower than that of the wireless link. As shown in Figure 12, we realize this scenario by setting the end-to-end bandwidth to 5, 10, and 15 Mbps at arbitrary times throughout the experiment. Leveraging the mechanism described in §4, BISCAY detects when the end-to-end bandwidth reduces compared to the KPI-based wireless link bandwidth estimate and swiftly switches to BBR. Conversely, when that condition is not met, BISCAY goes back to its normal operation mode. This prompt adaptation to changes in the network conditions ensures an effective utilization of the available bandwidth.

6.4 Wireless-aware CC Evaluation

So far, we have compared BISCAY with the most used and relevant CCAs; however, there are few wireless-specific CCAs that also leverage air-interface KPIs to operate that are missing



a) Biscay vs PBE-CC **b)** Biscay vs CUBIC and BBR

Figure 13 **a)** Throughput and delay comparison of Biscay compared with PBE-CC. **b)** Throughput and delay comparison of Biscay, BBR and CUBIC using a COTS UE (Google Pixel 5).

in the evaluation. Specifically, PBE-CC [101], a CCA built atop NG-Scope [100] which is LTE-only sniffer that extracts DCI messages from the air interface containing the scheduling grants of the users in a given cell. PBE-CC works on the principle of exploiting all the *available* PRBs that have not been allocated to any user in the air interface to transmit data. Note that PBE-CC operates only in 4G downlink direction, thus, we have implemented an equivalent version of Biscay for a fair comparison. Inspired by the feedback mechanism employed in prior wireless/cellular CCAs such as Sprout [97], in our implementation, the cellular downlink bandwidth derived from downlink KPIs (PRBs and TBS extracted from the DCI downlink grant) is sent to the sender using the TCP flow control mechanism. Biscay calculates the observed downlink bandwidth using OpenDiag and sends it to the other endpoint which uses it to determine how much traffic is sent.

Figure 13a demonstrates the comparison of Biscay and PBE-CC in commercial 4G network scenarios. We observe that both PBE-CC and Biscay achieve similar throughput as both maximize the available network bandwidth. However, Biscay outperforms PBE-CC in terms of delay with Biscay halving the average and tail delays compared to PBE-CC. This behavior can be attributed to PBE-CC's mechanism of increasing the downstream sending rate when PRBs are available and assuming that the base station will grant those resources to it. This simplistic and naive view of the base station's MAC scheduler is far from how commercial schedulers work. In practice, however, most if not all schedulers implement some variation of proportional fair (PF) which uses the number of UEs in the cell and the UE scheduling requests/buffer status reports (a reflection of how much data each UE wants to transmit), channel quality measurements (CQI and periodic measurement reports concluded by the UE and the base station), bearer quality of service and even the historic grant allocation as input. The performance of PBE-CC is intricately linked to the channel quality reports from both the User Equipment (UE) and the base station. The base station grants resources based on these quality assessments: a poor channel quality results in fewer PRBs with lower MCS to ensure the UE can decode them. While UEs typically receive resources if they have data to send, boast good channel quality, and haven't recently received resources. PBE-CC's narrow focus on only the available resources ignoring everything else can lead to queue buildups, thus increasing delays.

6.5 Biscay's Real-World Evaluation

Pantheon emulation can deviate from the real-world throughput and delays by up to 17% [104]. To demonstrate its deployability and to overcome the inherent limitations of emulation, we implemented Biscay on a COTS UE (Google Pixel 5). We evaluated it alongside BBR and CUBIC on real private and public networks. Real-world evaluations pose a significant challenge due to variable external conditions, like random noise and other users in the cell

across experiments. Such factors can significantly impact the performance of the different CCAs. To minimize such external factors across experiments, we used a private network with two base stations (leveraging srsRAN [88]) configured to be neighbor cells (with handovers between them) and core network (Open5GS [61]). To ensure reproducible network conditions, the experiments were conducted when there was no other COTS UE in the network. In addition to private network deployment, we also conducted evaluations on public networks of two major US operators (Verizon and T-Mobile). These evaluations span diverse scenarios, including: static/mobile, single-cell/CA, 4G/5G networks, and on-peak/off-peak periods. For consistency, the same mobility patterns and static positions were used across experiments in both private and public networks.

Figure 13b shows the results of experiments using a COTS UE with CUBIC, BBR and Biscay over both private and public networks. Similar to the results obtained with Pantheon, CUBIC saturates the bottleneck wireless link, resulting in maximum bandwidth usage and high average/tail delays due to queue build-up. Compared to CUBIC, BBR achieves similar average throughput, but has lower average and tail delays due to relatively more accurate bottleneck bandwidth estimation. Notably, Biscay outperforms both BBR and CUBIC in terms of throughput and delay in both private and public networks. Specifically, Biscay gets 4.6% higher throughput than CUBIC and BBR while reducing average and tail delays by 46% and 44%, respectively, compared to BBR. Interestingly, the highest delays experienced by Biscay are lower than the lowest delays experienced by CUBIC and BBR. This experiment demonstrates how the different components of the Biscay system effectively work together in challenging real-world conditions.

7 Conclusions

We propose Biscay, a practical and radio KPI-driven congestion control design for mobile networks. Biscay leverages OpenDiag, our in-kernel real-time radio KPI extraction tool that allows KPIs to be obtained from the radio modem at fine *ms* scale granularity. It enables Biscay to accurately determine the bottleneck bandwidth on the device side to achieve high throughput and low delays. Biscay is extensively evaluated and compared against 9 state-of-the-art CCAs in a wide variety of scenarios using our 4G/5G performance traces and real-world experiments using a commodity mobile device. Biscay shows a significant reduction in average and tail delays, notably 58%/41% and 98%/99% average/tail delay reduction compared with BBR and CUBIC, respectively.

References

1 5G NR User Equipment (UE) radio access capabilities. https://www.etsi.org/deliver/etsi_ts/138300_138399/138306/17.00.00_60/ts_138306v170000p.pdf.
2 eBPF. https://ebpf.io/.
3 FieldFox. https://www.keysight.com/zz/en/products/network-analyzers/fieldfox-handheld-rf-microwave-analyzers.html.
4 MobileInsight Core. https://github.com/mobile-insight/mobileinsight-core.
5 MobileInsight Mobile. https://github.com/mobile-insight/mobileinsight-mobile.
6 Powder. https://powderwireless.net/.
7 Qualcomm Hexagon architecture. https://developer.qualcomm.com/software/hexagon-dsp-sdk/dsp-processor.
8 Qualcomm QMI (Qualcomm MSM Interface). https://osmocom.org/projects/quectel-modems/wiki/QMI.
9 Radio Interface Layer (RIL). https://source.android.com/docs/core/connect/ril.

10 Samsung Exynos drivers (Android kernel). https://android.googlesource.com/kernel/exynos/+/refs/tags/android-5.1.1_r0.26/drivers/.

11 3GPP. 5G NR Medium Access Control (MAC) protocol specification. https://www.etsi.org/deliver/etsi_ts/138300_138399/138321/15.03.00_60/ts_138321v150300p.pdf.

12 3GPP. 5G NR Non-Access-Stratum (NAS) protocol for 5G System (5GS). https://www.etsi.org/deliver/etsi_ts/124500_124599/124501/16.05.01_60/ts_124501v160501p.pdf.

13 3GPP. 5G NR Packet Data Convergence Protocol (PDCP) specification. https://www.etsi.org/deliver/etsi_ts/138300_138399/138323/16.02.00_60/ts_138323v160200p.pdf.

14 3GPP. 5G NR Physical layer: General description . https://www.etsi.org/deliver/etsi_ts/138200_138299/138201/15.00.00_60/ts_138201v150000p.pdf.

15 3GPP. 5G NR Physical Layer Procedures for Control. https://www.etsi.org/deliver/etsi_ts/138200_138299/138213/16.02.00_60/ts_138213v160200p.pdf.

16 3GPP. 5G NR Physical Layer Procedures for Data. https://www.etsi.org/deliver/etsi_ts/138200_138299/138214/16.02.00_60/ts_138214v160200p.pdf.

17 3GPP. 5G NR Radio Link Control (RLC) protocol specification. https://www.etsi.org/deliver/etsi_ts/138300_138399/138322/16.01.00_60/ts_138322v160100p.pdf.

18 3GPP. 5G NR Radio Resource Control (RRC) protocol specification. https://www.etsi.org/deliver/etsi_ts/138300_138399/138331/16.01.00_60/ts_138331v160100p.pdf.

19 3GPP. 5G NR Service Data Adaptation Protocol (SDAP) specification. https://www.etsi.org/deliver/etsi_ts/137300_137399/137324/16.02.00_60/ts_137324v160200p.pdf.

20 3GPP. 5G NR User Equipment (UE) radio transmission and reception. https://www.etsi.org/deliver/etsi_ts/138100_138199/13810102/15.04.00_60/ts_13810102v150400p.pdf.

21 3GPP. LTE E-UTRA Physical Layer Procedures. https://www.etsi.org/deliver/etsi_ts/136200_136299/136213/14.02.00_60/ts_136213v140200p.pdf.

22 3GPP. LTE Radio measurement collection for Minimization of Drive Tests. https://www.etsi.org/deliver/etsi_ts/137300_137399/137320/16.02.00_60/ts_137320v160200p.pdf.

23 Soheil Abbasloo, Chen-Yu Yen, and H. Jonathan Chao. Classic meets modern: A pragmatic learning-based congestion control for the internet. SIGCOMM '20, page 632–647, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3387514.3405892.

24 Accuver. XCAL-Mobile. https://accuver.com/sub/products/view.php?idx=10.

25 Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, page 63–74, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1851182.1851192.

26 Thomas Anderson, Andrew Collins, Arvind Krishnamurthy, and John Zahorjan. PCP: Efficient endpoint congestion control. In *Proceedings of the 3rd Conference on Networked Systems Design & Implementation - Volume 3*, NSDI'06, page 15, USA, 2006. USENIX Association.

27 Anon. Apple IOS baseband commands. https://www.theiphonewiki.com/wiki/Talk:Baseband_Commands.

28 Anon. qcombbdbg. https://code.google.com/archive/p/qcombbdbg/.

29 Venkat Arun and Hari Balakrishnan. Copa: Practical Delay-Based congestion control for the internet. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 329–342, Renton, WA, Apr 2018. USENIX Association. URL: https://www.usenix.org/conference/nsdi18/presentation/arun.

30 The Computer Security Group at Berlin University of Technology. SCAT: Signaling Collection and Analysis Tool. https://github.com/fgsect/scat.

31 Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. *SIGCOMM Comput. Commun. Rev.*, 24(4):24–35, Oct 1994. doi:10.1145/190809.190317.

32 Joe Breen, Andrew Buffmire, Jonathon Duerig, Kevin Dutt, Eric Eide, Mike Hibler, David Johnson, Sneha Kumar Kasera, Earl Lewis, Dustin Maas, Alex Orange, Neal Patwari, Daniel Reading, Robert Ricci, David Schurig, Leigh B. Stoller, Jacobus Van der Merwe, Kirk Webb, and Gary Wong. POWDER: Platform for open wireless data-driven experimental research. In *Proceedings of the 14th International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization*, WiNTECH'20, page 17–24, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3411276.3412204.

33 Nicola Bui and Joerg Widmer. OWL: A reliable online watcher for LTE control channel measurements. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, ATC '16, page 25–30, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2980055.2980057.

34 Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR: Congestion-based congestion control. *Commun. ACM*, 60(2):58–66, Jan 2017. doi:10.1145/3009824.

35 Federico Chiariotti, Andrea Zanella, Stepan Kucera, and Holger Claussen. Bbr-s: A low-latency bbr modification for fast-varying connections. *IEEE Access*, 9:76364–76378, 2021. doi:10.1109/ACCESS.2021.3082715.

36 Mo Dong, Qingxi Li, Doron Zarchy, P. Brighten Godfrey, and Michael Schapira. PCC: Re-architecting congestion control for consistent high performance. NSDI'15, page 395–408, USA, 2015. USENIX Association.

37 Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, P. Brighten Godfrey, and Michael Schapira. PCC Vivace: Online-learning congestion control. In *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation*, NSDI'18, page 343–356, USA, 2018. USENIX Association.

38 T Engel. Intel xgoldmon driver. https://github.com/2b-as/xgoldmon.

39 Ericsson. Mobile Subscriptions Outlook. https://www.ericsson.com/en/reports-and-papers/mobility-report/dataforecasts/mobile-subscriptions-outlook.

40 Alisa Esage. Advance Hexagon DIAG architecture. https://media.ccc.de/v/rc3-11397-advanced_hexagon_diag.

41 Robert Falkenberg and Christian Wietfeld. FALCON: An accurate real-time monitor for client-based mobile network data analytics. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7, 2019. doi:10.1109/GLOBECOM38437.2019.9014096.

42 Endless OS Foundation. ModemManager. https://github.com/endlessm/ModemManager.

43 Moinak Ghoshal, Imran Khan, Z. Jonny Kong, Phuc Dinh, Jiayi Meng, Y. Charlie Hu, and Dimitrios Koutsonikolas. Performance of cellular networks on the wheels. In *Proceedings of the 2023 ACM on Internet Measurement Conference*, IMC '23, page 678–695, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3618257.3624814.

44 Google. Android Telephony API. https://developer.android.com/reference/android/telephony/package-summary.

45 Prateesh Goyal, Anup Agarwal, Ravi Netravali, Mohammad Alizadeh, and Hari Balakrishnan. ABC: A simple explicit congestion controller for wireless networks. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 353–372, Santa Clara, CA, Feb 2020. USENIX Association. URL: https://www.usenix.org/conference/nsdi20/presentation/goyal.

46 Prateesh Goyal, Mohammad Alizadeh, and Hari Balakrishnan. Rethinking congestion control for cellular networks. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, HotNets-XVI, page 29–35, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3152434.3152437.

47 A. Gurtov, T. Henderson, and S. Floyd. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 3782, RFC Editor, Apr 2004. URL: https://www.rfc-editor.org/rfc/rfc3782.txt.

48 Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: A new TCP-friendly high-speed TCP variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, Jul 2008. doi:10.1145/1400097.1400105.

49 V. Jacobson. Congestion avoidance and control. In *Symposium Proceedings on Communications Architectures and Protocols*, SIGCOMM '88, page 314–329, New York, NY, USA, 1988. Association for Computing Machinery. doi:10.1145/52324.52356.

50 Rajendra K Jain, Dah-Ming W Chiu, William R Hawe, et al. A quantitative measure of fairness and discrimination. *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, 21, 1984.

51 Keysight. Nemo Analyze. https://www.keysight.com/us/en/product/NTN00000B/nemo-analyze-drive-test-post-processing-solution.html.

52 Keysight. Nemo Handy. https://www.keysight.com/us/en/product/NTH00000B/nemo-handy-handheld-measurement-solution.html.

53 I. Khan, M. Ghoshal, J. Angjo, S. Dimce, M. Hussain, P. Parastar, Y. Yu, C. Fiandrino, C. Orfanidis, S. Aggarwal, A. Aguiar, O. Alay, C. Chiasserini, F. Dressler, Y. Hu, S. Ko, D. Koutsonikolas, and J. Widmer. How mature is 5g deployment? a cross-sectional, year-long study of 5g uplink performance. In *IFIP Networking*, pages –, June 2024.

54 Imran Khan, Tuyen X. Tran, Matti Hiltunen, Theodore Karagioules, and Dimitrios Koutsonikolas. An experimental study of low-latency video streaming over 5g, 2024. arXiv:2403.00752.

55 Morteza Kheirkhah, Mohamed M. Kassem, Gorry Fairhurst, and Mahesh K. Marina. XRC: An explicit rate control for future cellular networks. In *ICC 2022 - IEEE International Conference on Communications*, pages 2290–2296, 2022. doi:10.1109/ICC45855.2022.9839150.

56 Jaehong Kim, Yunheon Lee, Hwijoon Lim, Youngmok Jung, Song Min, and Kim Dongsu Han. OutRAN: Co-optimizing for flow completion time in radio access network. In *Proceedings of the 18th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '22, page 369–385, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3555050.3569122.

57 Leonard Kleinrock. Internet congestion control using the power metric: Keep the pipe just full, but no fuller. *Ad Hoc Networks*, 80:142–157, 2018. URL: https://www.sciencedirect.com/science/article/pii/S1570870518302476, doi:10.1016/j.adhoc.2018.05.015.

58 Swarun Kumar, Ezzeldin Hamed, Dina Katabi, and Li Erran Li. LTE radio analytics made easy and accessible. *SIGCOMM Comput. Commun. Rev.*, 44(4):211–222, Aug 2014. doi:10.1145/2740070.2626320.

59 Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, and Zhongyi Shi. The quic transport protocol: Design and internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 183–196, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3098822.3098842.

60 Jinsung Lee, Sungyong Lee, Jongyun Lee, Sandesh Dhawaskar Sathyanarayana, Hyoyoung Lim, Jihoon Lee, Xiaoqing Zhu, Sangeeta Ramakrishnan, Dirk Grunwald, Kyunghan Lee, and Sangtae Ha. PERCEIVE: Deep learning-based cellular uplink prediction using real-time scheduling patterns. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, MobiSys '20, page 377–390, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3386901.3388911.

61 Sukchan Lee. Open5GS. https://open5gs.org/.

62 Wai Kay Leong, Zixiao Wang, and Ben Leong. TCP congestion control beyond bandwidth-delay product for mobile cellular networks. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '17, page 167–179, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3143361.3143378.

63 Yuanjie Li, Chunyi Peng, Zengwen Yuan, Jiayao Li, Haotian Deng, and Tao Wang. MobileInsight: Extracting and analyzing cellular network information on smartphones. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, MobiCom '16, page 202–215, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2973750.2973751.

64 Yuanjie Li, Chunyi Peng, Zhehui Zhang, Zhaowei Tan, Haotian Deng, Jinghao Zhao, Qianru Li, Yunqi Guo, Kai Ling, Boyan Ding, Hewu Li, and Songwu Lu. Experience: A five-year retrospective of MobileInsight. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, MobiCom '21, page 28–41, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3447993.3448138.

65 Shao Liu, Tamer Başar, and R. Srikant. TCP-Illinois: A loss and delay-based congestion control algorithm for high-speed networks. valuetools '06, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1190095.1190166.

66 Feng Lu, Hao Du, Ankur Jain, Geoffrey M. Voelker, Alex C. Snoeren, and Andreas Terzis. CQIC: Revisiting cross-layer congestion control for cellular networks. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, HotMobile '15, page 45–50, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2699343.2699345.

67 MediaTek. MediaTek diagnostic driver (Android kernel). https://android.googlesource.com/kernel/mediatek/+/refs/tags/android-6.0.1_r0.110/drivers/misc/mediatek/.

68 moiji mobile.com. diag-parser. https://github.com/moiji-mobile/diag-parser.

69 Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate record-and-replay for HTTP. In *Usenix annual technical conference*, pages 417–429, 2015.

70 Kathleen Nichols and Van Jacobson. Controlling queue delay. *Commun. ACM*, 55(7):42–50, Jul 2012. doi:10.1145/2209249.2209264.

71 openqst. openpst. https://github.com/openpst/openpst.

72 Osmocom. Osmocom Diag documentation. https://osmocom.org/projects/quectel-modems/wiki/Diag.

73 P1Sec. QCSuper. https://github.com/P1sec/QCSuper.

74 Shinik Park, Jinsung Lee, Junseon Kim, Jihoon Lee, Sangtae Ha, and Kyunghan Lee. ExLL: An extremely low-latency congestion control for mobile cellular networks. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '18, page 307–319, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3281411.3281430.

75 Qtrun. NetworkSignalGuru. https://m.qtrun.com/en/product.html.

76 Qualcomm. Qualcomm Diag driver (Android kernel). https://android.googlesource.com/kernel/msm.git/+/refs/tags/android-11.0.0_r0.27/drivers/char/diag/.

77 Qualcomm. Qualcomm QXDM. https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/80-n9471-1_d_qxdm_professional_tool_quick_start.pdf.

78 Ahmad Rahmati, Clayton Shepard, Chad C. Tossell, Lin Zhong, Philip Kortum, Angela Nicoara, and Jatinder Singh. Seamless tcp migration on smartphones without network support. *IEEE Transactions on Mobile Computing*, 13(3):678–692, 2014. doi:10.1109/TMC.2013.8.

79 Rohde & Schwarz. QualiPoc Android. https://www.rohde-schwarz.com/us/product/qualipoc_android-productstartpage_63493-55430.html.

80 Rohde & Schwarz. TSMA6. https://www.mobilewirelesstesting.com/wp-content/uploads/2019/02/TSMA6_bro_5215-8241-12_v0400.pdf.

81 William Sentosa, Balakrishnan Chandrasekaran, P. Brighten Godfrey, Haitham Hassanieh, and Bruce Maggs. DChannel: Accelerating mobile applications with parallel high-bandwidth and low-latency channels. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 419–436, Boston, MA, Apr 2023. USENIX Association. URL: https://www.usenix.org/conference/nsdi23/presentation/sentosa.

82 Stanislav Shalunov, Greg Hazel, Jana Iyengar, and Mirja Kühlewind. Low Extra Delay Background Transport (LEDBAT). RFC 6817, Dec 2012. URL: https://www.rfc-editor.org/info/rfc6817, doi:10.17487/RFC6817.

83 Benjamin Sliwa, Hendrik Schippers, and Christian Wietfeld. Machine learning-enabled data rate prediction for 5G NSA vehicle-to-cloud communications. In *2021 IEEE 4th 5G World Forum (5GWF)*, pages 299–304, 2021. doi:10.1109/5GWF52925.2021.00059.

84 Benjamin Sliwa and Christian Wietfeld. Data-driven network simulation for performance analysis of anticipatory vehicular communication systems. *IEEE Access*, 7:172638–172653, 2019. doi:10.1109/ACCESS.2019.2956211.

85 Gyokov Solutions. G-NetTrack Pro. http://www.gyokovsolutions.com/.

86 SRLabs. SnoopSnich. https://opensource.srlabs.de/projects/snoopsnitch.

87 SRS. AirScope. https://www.srs.io/products/#SRS-Airscope.

88 Software Radio Systems. srsRAN. https://www.srsran.com/.

89 K. Tan, J. Song, Q. Zhang, and M. Sridharan. A compound TCP approach for high-speed and long distance networks. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pages 1–12, 2006. doi:10.1109/INFOCOM.2006.188.

90 Zhaowei Tan, Jinghao Zhao, Boyan Ding, and Songwu Lu. CellDAM: User-Sp ace, rootless detection and mitigation for 5g data plane. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1601–1619, Boston, MA, Apr 2023. USENIX Association. URL: https://www.usenix.org/conference/nsdi23/presentation/tan.

91 Zhaowei Tan, Jinghao Zhao, Yuanjie Li, Yifei Xu, Yunqi Guo, and Songwu Lu. Ldrp: Device-centric latency diagnostic and reduction for cellular networks without root. *IEEE Transactions on Mobile Computing*, 23(4):2748–2764, 2024. doi:10.1109/TMC.2023.3267805.

92 MI Team. MobileINsight website. http://www.mobileinsight.net/.

93 Google TCP/QUIC BBR teams. BBRv3: Algorithm Bug Fixes and Public Internet Deployment. https://datatracker.ietf.org/meeting/117/materials/slides-117-ccwg-bbrv3-algorithm-bug-fixes-and-public-internet-deployment-00.

94 Han Tian, Xudong Liao, Chaoliang Zeng, Junxue Zhang, and Kai Chen. Spine: An efficient drl-based congestion control with ultra-low overhead. In *Proceedings of the 18th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '22, page 261–275, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3555050.3569125.

95 David X. Wei, Cheng Jin, Steven H. Low, and Sanjay Hegde. FAST TCP: Motivation, architecture, algorithms, performance. *IEEE/ACM Transactions on Networking*, 14(6):1246–1259, 2006. doi:10.1109/TNET.2006.886335.

96 Keith Winstein and Hari Balakrishnan. TCP Ex Machina: Computer-generated congestion control. *SIGCOMM Comput. Commun. Rev.*, 43(4):123–134, Aug 2013. doi:10.1145/2534169.2486020.

97 Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, page 459–472, USA, 2013. USENIX Association.

98 Xiufeng Xie, Xinyu Zhang, Swarun Kumar, and Li Erran Li. PiStream: Physical layer informed adaptive video streaming over LTE. *GetMobile: Mobile Comp. and Comm.*, 20(2):31–34, Oct 2016. doi:10.1145/3009808.3009819.

99 Xiufeng Xie, Xinyu Zhang, and Shilin Zhu. Accelerating mobile web loading using cellular link information. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '17, page 427–439, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3081333.3081367.

100 Yaxiong Xie and Kyle Jamieson. NG-Scope: Fine-grained telemetry for nextg cellular networks. *Proc. ACM Meas. Anal. Comput. Syst.*, 6(1), Feb 2022. doi:10.1145/3508032.

101 Yaxiong Xie, Fan Yi, and Kyle Jamieson. PBE-CC: Congestion control via endpoint-centric, physical-layer bandwidth measurements. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 451–464, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3387514.3405880.

102 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 Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 479–494, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3387514.3405882.

103 Qiang Xu, Sanjeev Mehrotra, Zhuoqing Mao, and Jin Li. PROTEUS: Network performance forecast for real-time, interactive mobile applications. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, page 347–360, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2462456.2464453.

104 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, Boston, MA, Jul 2018. USENIX Association. URL: https://www.usenix.org/conference/atc18/presentation/yan-francis.

105 Yasir Zaki, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. Adaptive congestion control for unpredictable cellular networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 509–522, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2785956.2787498.

106 Shengtong Zhu, Yan Liu, Lingfeng Guo, Rudolf K. H. Ngan, and Jack Y. B. Lee. mbbr - improving bbr performance over rate-limited mobile networks. In *2023 IEEE 31st International Conference on Network Protocols (ICNP)*, pages 1–11, 2023. doi:10.1109/ICNP59255.2023.10355605.

A Appendix

A.1 Dataset and evaluation scenarios

The result of our measurement campaign is a dataset that contains 4G/5G traces in equal proportion, collected in a variety of cities across Europe and the US. Within the 4G portion, 50% of the traces are collected in static scenarios and 50% under mobility. However, the 5G part of the dataset is split in 20% static and 80% mobility. This distribution may lead to artifacts in the evaluation, such as CCAs appearing to perform better under mobility scenarios. This outcome might seem counterintuitive given that 5G generally offers superior performance, with higher throughput and lower latency. Across both 4G and 5G traces, 60% were collected during peak hours, while the remaining 40% during off-peak hours. In addition to those four scenarios, the collected traces as well as the real-world experiments capture the complete range in terms of connection states and events [18]: 4G/5G RRC idle to connected (the modem is disconnected from the network and new connection is established in order to send traffic), 5G RRC inactive to connected (the modem is not fully disconnected from the network and the previous RRC session gets reused to establish a data channel), network attachment and sessions establishments over all the radio access technologies (4G/5G NSA/5G SA) and all possible handover combinations (4G-to-4G, 4G-to-5G, 5G-to-4G and 5G-to-5G). All the previously listed events and state transitions are evenly distributed across the different scenarios captured in the measurement campaign as well as the live experiments.

```
# This callback is triggered when:
# - Receive ACK
# - Timeout
# - Duplicate ACK
# - Explicit Congestion Notification (ECN)
biscay_congestion_window_callback():
    # Transition between states based on network conditions
    if state == STARTUP:
        cwnd = SlowStart()
        # Try to get bandwidth from KPIs
        if getCellularBW() == OK:
            # Swicth to Biscay when we get KPI-based bandwidth
            state = BISCAY

    # Biscay mode: the bottleneck is in the RAN
    elif state == BISCAY:
        # Get cellular bandwidth from KPIs (OpenDiag)
        cellular_bw = getCellularBW()
        # Get end-to-end bandwidth (BBR)
        end_to_end_bw = getEndToEndBandiwdth(BBR)
        # Get RTT
        rtt = min(rtt, MeasureRTT(last_ack))
        # Set CWND
        cwnd = BandwidthDelayProduct(BWSplitPolicy(cellular_bw), rtt)
        # Check bottleneck location
        if cellular_bw > end_to_end_bw: # Bottleneck in the wired segment
            # Trigger fallback mechanism
            state = FALLBACK
            # Set CWND using the wired-specific CCA selected (BBR)
            cwnd = setCWNDfromFallbackCCA(BBR)

    # Fallback mode: the bottleneck is in the wired segment
    elif state == FALLBACK:
        # Set CWND using the wired-specific CCA selected (BBR)
        cwnd = setCWNDfromFallbackCCA(BBR)

        # Get cellular bandwidth from KPIs (OpenDiag)
        cellular_bw = getCellularBW()
        # Get end-to-end bandwidth (BBR)
        end_to_end_bw = getEndToEndBandiwdth(BBR)
        # Check if the bottleneck has changed
        if cellular_bw == end_to_end_bw: # Bottleneck in the cellular link
            # Disable fallback mechanism
            state = BISCAY
            # Get RTT
            rtt = min(rtt, MeasureRTT(last_ack))
            # Set CWND
            cwnd = BandwidthDelayProduct(BWSplitPolicy(cellular_bw), rtt)
    return cwnd

# Function that splits the available bandwidth equally for all flows
BWSplitPolicy(bw):
    # Get number of active flows (Including UDP)
    num_flows = getNumberActiveFlows()
    return bw/num_flows
```

A.2 Discussion on Biscay CC internals

A.2.1 Biscay and BBR

We present the pseudo-code for Biscay's functionality. Like other CC algorithms in Linux, Biscay's CC module gets deployed in the kernel as a module and the congestion control function (*calculate_congestion_window_callback()*) is invoked as a callback. Biscay logical structure is a state machine with three states: `STARTUP, BISCAY`, and `FALLBACK`.

`STARTUP`: When a TCP socket is open, it goes into the `STARTUP` phase. We leverage slow-start [49], which exponentially increases the congestion window in order to fill the pipe quickly. However, unlike other CCAs that remain in the slow-start phase until a congestion condition is met (e.g., the bandwidth reaches a ceiling, packets start dropping, delay increases, etc.), Biscay exits the `STARTUP` state and moves into `BISCAY` when a reliable KPI-based bandwidth prediction can be made (usually after a couple of slow-start iterations). In practice, when the phone has been running for some time, this phase is mostly skipped, given that it is highly probable that when a socket gets open and starts transmitting (`STARTUP` state), other sockets are already transmitting in the system and therefore reliable KPI-based bandwidth predictions from OpenDiag can be obtained.

`BISCAY`: During the `BISCAY`, Biscay sets the congestion window according to the logic defined in §4. The cellular bandwidth (*cellular_bw*) is calculated from the KPIs extracted with OpenDiag and combined after splitting it according to the number of active flows (*BWSplitPolicy(bw)*) with the RTT (obtained as the minimum of the previous RTT value and the RTT of the last ACK) into a congestion window value using the BDP. Then, Biscay checks for a change in the bottleneck location. It does so by comparing the end-to-end bandwidth with the cellular bandwidth. In case the bottleneck moves to the wired segment, Biscay switches state to `FALLBACK` and falls back to an end-to-end CCA (BBR). Otherwise, it returns the congestion window previously calculated.

`FALLBACK`: In the `FALLBACK`, Biscay calculates the congestion window using the selected wired-specific CCA (BBR). Then, similar to `BISCAY` state, Biscay checks for changes in the location of the bottleneck using the cellular bandwidth and the end-to-end bandwidth. If the bottleneck switches, the state changes to `BISCAY`.

A.2.2 Multi-flow bandwidth distribution

Given the flow's homogeneity [78] (where the average lifespan of flows tends to be similar), we have implemented a simple yet effective bandwidth distribution policy that equally splits the available bandwidth among the active flows. This approach not only promotes fairness but also ensures efficiency by avoiding complex kernel-level computations, particularly floating-point operations, which are unsupported and could introduce additional delays. Although Biscay's focus is on TCP, our bandwidth distribution policy also aims for fairness with other transport protocols by accounting for UDP connections (QUIC [59], which is used by a number of Google applications and web-based applications, runs over UDP). It does so by also including UDP active flows when calculating the number of active flows in the system (*getNumberActiveFlows()*), ensuring fairness across transport protocols. However, there are still some scenarios that are negatively affected by our bandwidth distribution policy. The most relevant one is an application that opens multiple TCP sockets. It would have an advantage over an application that only uses one socket. This could be addressed with an iteration of our policy which, rather than targeting inter-flow fairness, uses inter-app fairness. This could be achieved by looking at the process ID (PID) of each active flow and proportionally assigning bandwidth to the PID rather than the flows open by the process.

Alternatively, more advanced scheduling mechanisms, such as round-robin or proportional fairness, could be integrated to address these limitations.

A.2.3 Biscay and other CCA

Inspired by newer versions of BBR (v2 and v3) that integrate additional signals for enhanced end-to-end bandwidth estimation [93], Biscay could also leverage congestion signals from other CCAs. From CUBIC and Reno, Biscay could use packet loss to complement the end-to-end bandwidth determination method and make it more precise. CUBIC/Reno could be used as a fallback CCA in case Biscay detects that the bottleneck has shifted to the wired segment. Similarly, Explicit Congestion Notification (ECN) can complement the end-to-end metric. Other domain-specific CCAs, such as DCTCP [25], rely on queue occupancy and packet reordering to determine when the bottleneck is reached. Queue occupancy metrics could improve Biscay 's cellular bandwidth estimation by leveraging RLC-layer queue sizes obtained directly from the modem via OpenDiag. Using techniques from TCP Vegas and Compound TCP, Biscay could give more importance to time-based metrics such as one-way delay (delay gradient) or jitter to improve its bandwidth localization accuracy.