RATO: RAN-Assisted Transport Optimization for 5G and Beyond

BIANNUAL PROGRESS SEMINAR REPORT

Submitted by:
Arghyadip Chakraborty

Guided by:
Prof. Mythili Vutukuru

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Indian Institute of Technology Bombay

July 2026

Abstract

The fifth generation (5G) of mobile communication systems promises high data rates and low latency for demanding applications, yet fully realizing this potential requires efficient end-to-end transport. Modern transport protocols regulate their transmission rates by relying on end-to-end congestion signals such as packet loss, delivery rate, and round-trip time (RTT). However, the highly dynamic nature of the Radio Access Network (RAN)—driven by fluctuating channel conditions, scheduling decisions, and retransmissions—often causes these protocols to misinterpret transient wireless effects as persistent network congestion. This limited visibility into the radio layer results in suboptimal transmission decisions, which can lead to excessive queue buildup, increased latency, and inefficient utilization of available radio resources.

To address this fundamental visibility gap, this report introduces **RAN-Assisted Transport Optimization (RATO)**, a framework designed to enable transport-aware telemetry and control within the 5G RAN. Built upon the Janus programmable networking platform, RATO utilizes user-space eBPF codelets embedded directly within the RAN stack to expose real-time, transport-relevant metrics. By securely exporting internal data such as queue occupancy, queueing delay, and dequeue rate, RATO bridges the gap between end-to-end transport protocols and underlying wireless network dynamics. This cross-layer observability allows the RAN state to be translated into actionable transport-layer signals without requiring invasive, hardcoded modifications to the network implementation.

The capabilities of the RATO framework are evaluated using a comprehensive end-to-end 5G testbed, featuring a proof-of-concept Explicit Congestion Notification (ECN)/L4S marking mechanism. This mechanism tracks packets and estimates queue states to generate congestion signals that accurately reflect current radio access conditions. Experimental results confirm a strong correlation between RATO's exported telemetry and actual transport behavior. Most notably, when applied to ECN-aware congestion control algorithms like TCP Cubic, RATO's marking mechanism achieved nearly an order-of-magnitude reduction in latency, dropping RTT from 400-500 ms to 40-50 ms. These performance gains are achieved while maintaining comparable throughput and minimizing retransmissions, demonstrating the profound benefits of integrating programmable, real-time RAN telemetry into transport-layer decision-making.

Contents

1 Introduction

The fifth generation (5G) of mobile communication systems is designed to provide high data rates, low latency, improved reliability, and support for a massive number of connected devices. These capabilities enable a wide range of applications, including cloud gaming, virtual and augmented reality, industrial automation, and real-time analytics, all of which place increasingly stringent demands on network performance.

Achieving the full potential of 5G requires not only advances in radio access and core network infrastructure, but also efficient end-to-end transport. Modern transport protocols regulate their transmission rates using congestion control algorithms, which continuously adapt to changing network conditions in an effort to balance throughput, latency, and packet loss. Different congestion control algorithms rely on different congestion signals, including packet loss, round-trip time (RTT), delivery rate, and Explicit Congestion Notification (ECN) marks.

However, the Radio Access Network (RAN) exhibits highly dynamic behavior due to fluctuating channel conditions, scheduling decisions, retransmissions, and varying radio resource allocations. These factors can rapidly change the bandwidth and latency experienced by a flow, even in the absence of persistent network congestion. Since transport protocols observe the network only through end-to-end signals, they have limited visibility into the underlying radio-layer conditions that drive these performance variations.

As a result, congestion control algorithms may misinterpret transient wireless effects as congestion or fail to react when congestion is actually developing. This can lead to unnecessary rate reductions, excessive queue buildup, increased latency, unstable throughput, and inefficient utilization of available radio resources.

These challenges motivate tighter interaction between the transport layer and the RAN. Since all user traffic traverses the RAN and the RAN possesses real-time knowledge of wireless network conditions, it represents a natural point for collecting telemetry and influencing transport behavior. This observation forms the basis of RATO (RAN-Assisted Transport Optimization), which leverages programmable telemetry and control within the RAN to expose transport-relevant information and enable transport-layer optimization in 5G networks.

1.1 5G Architecture

A 5G network consists of three primary components: the User Equipment (UE), the Radio Access Network (RAN), and the 5G Core (5GC). The UE, such as a smartphone, modem, or IoT device, connects wirelessly to a base station known as the gNodeB (gNB). The gNB forms part of the RAN and is responsible for radio resource management, scheduling, mobility handling, and communication with the core network.

The 5G Core provides functions such as authentication, mobility management, session management, and packet routing. Through the User Plane Function (UPF), user traffic is forwarded between the RAN and external data networks, including Internet services and cloud applica-

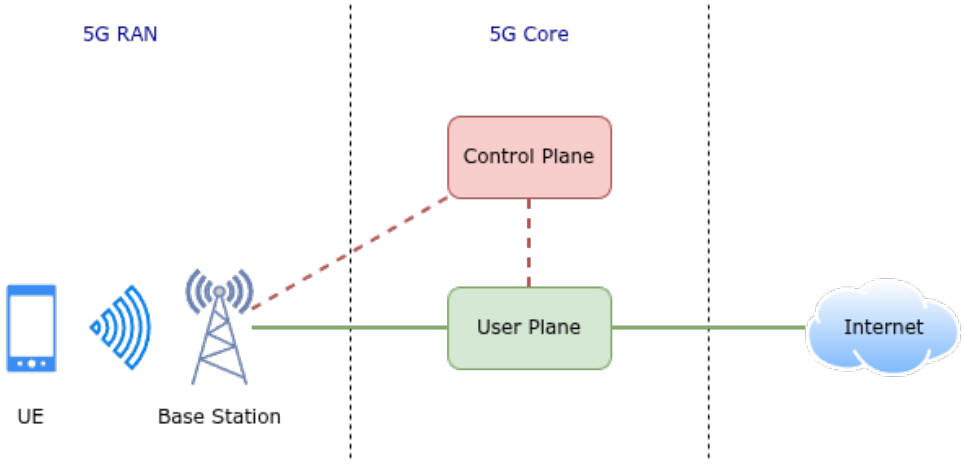

Figure 1.1: 5G Architecture

tions. Together, these components provide end-to-end connectivity between users and remote servers.

Figure 1.1 illustrates the high-level architecture of a 5G network and the interaction between the UE, RAN, 5G Core, and the Internet.

1.2 Transport over 5G

Figure 1.2 illustrates the end-to-end transport datapath through a 5G network. Traffic generated by an application traverses the Internet and the 5G Core before entering the gNB, where it passes through multiple protocol layers prior to transmission over the air interface.

Unlike conventional wired networks, the RAN performs a wide range of operations that directly influence packet delivery. These include packet buffering, segmentation and reassembly, retransmission of lost data, scheduling of radio resources, adaptive modulation and coding, and dynamic allocation of wireless capacity among users. As a result, the bandwidth, latency, and queueing behavior experienced by a flow can vary significantly over short timescales.

Many of these operations are internal to the RAN and are not directly visible to transport protocols. Instead, transport protocols must infer network conditions indirectly through end-to-end signals such as packet loss, round-trip time (RTT), delivery rate, and Explicit Congestion

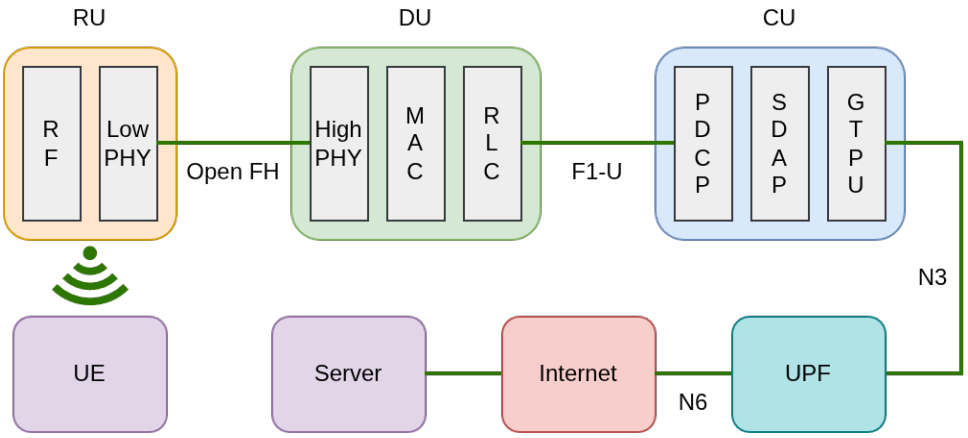

Figure 1.2: Transport datapath over 5G

Notification (ECN) marks. This limited visibility can lead to inaccurate congestion estimation and suboptimal transmission decisions, resulting in reduced throughput, increased latency, and inefficient utilization of available radio resources.

Since all user traffic traverses the RAN and the RAN possesses real-time knowledge of these internal dynamics, it represents a natural point for collecting telemetry and influencing transport behavior. This observation motivates RAN-Assisted Transport Optimization (RATO), which leverages programmable telemetry and control mechanisms within the RAN to expose transport-relevant information and improve transport-layer performance.

1.3 Contributions

In this work, we propose **RATO (RAN-Assisted Transport Optimization)**, a framework built on top of the Janus programmable networking platform that enables transport-aware telemetry and control within the 5G Radio Access Network (RAN). By leveraging programmable eBPF codelets embedded directly within the RAN, RATO bridges the gap between transport protocols and the underlying wireless network, exposing information that is otherwise unavailable to end hosts.

RATO provides:

- **Real-time RAN telemetry**, exporting transport-relevant metrics such as queue occupancy, queueing delay, and dequeue rate from within the RAN.
- **Cross-layer observability**, enabling direct correlation between transport-layer behavior and internal RAN dynamics through integrated transport and RAN monitoring.
- **Programmable transport control**, allowing RAN state to be translated into actionable transport-layer signals, demonstrated through a proof-of-concept ECN/L4S marking mechanism.
- **An experimental evaluation platform**, comprising a complete 5G testbed and visualization pipeline for developing and evaluating RAN-assisted transport optimization techniques.

2 Related Work

The optimization of transport layer performance in 5G and beyond networks necessitates a holistic approach that bridges the fundamental divide between end-to-end protocol logic and real-time Radio Access Network (RAN) dynamics. As high-speed wireless environments introduce unprecedented channel volatility, the research landscape has expanded from traditional endpoint refinements to deep architectural integrations. This chapter examines the evolution of these strategies, tracing the progress from host-based algorithmic modifications and intelligent RAN controllers to high-fidelity congestion signaling and the foundational frameworks required for programmable user-plane observability.

2.1 Algorithmic Enhancements

To address the inherent limitations of standard transport protocols in high-volatility environments, several researchers have proposed end-to-end algorithmic modifications designed to better interpret 5G channel dynamics. Saleh M. Abdullah et al. [1] proposed an **Enhanced NewReno** mechanism specifically for 5G environments. The basic idea is to move away from the "blind" linear growth of standard NewReno by estimating real-time data transfer rates. The implementation enables the sender to monitor the ACK stream and utilize dynamic thresholds to either double the window growth when capacity is abundant or freeze it immediately when the network is overburdened. Similarly, Kanagarathinam et al. [12] introduced **NexGen D-TCP**, which uses a discrete-time filter based on Tustin approximation to extract network parameters such as traffic intensity. This sender-side implementation adaptively adjusts the congestion window ($cwnd$) based on a control factor derived from estimated bandwidth.

To address RTT-related challenges in mmWave, Alramli et al. [4]**MSS-TCP** [3] developed **RTTV-TCP** and . These works utilize the variance between base, current, and maximum RTTs to calculate dynamic increment factors for the $cwnd$. By using a square-root weight factor at the transport layer, the implementation creates a convex-up growth curve that ensures the "pipe" remains full during rapid Line-of-Sight (LoS) to Non-Line-of-Sight (NLoS) transitions. In the same domain, Prause and Akselrod [27] introduced **TCP ROCCET**, an RTT-oriented extension for CUBIC. It mitigates bufferbloat by adding a latency-aware layer to CUBIC's loss-based logic, implemented as a pluggable Linux kernel module that manages the connection through distinct LAUNCH and ORBITER phases.

Protocol fairness and startup efficiency have also seen significant research. Jinlin Xu et al. [37] proposed **BBR-PG (Pacing Gain)**, which addresses the throughput discrepancy in flows with disparate RTTs by replacing BBR's fixed pacing gains with an adaptive regulatory factor. The **Yinker** algorithm by Xie et al. [36] further enhances BBR flexibility by using a Sigmoid function to interpret RTT variations at the sender side, dynamically adjusting the pacing rate based on measured packet loss. To optimize the initial connection phase, Guo et al. [10] developed **Stateful-BBR**, which leverages the "state" of previous flows to the same peer. The implementation maintains a stateful metrics table at the server to cache bandwidth and RTT, allowing new

flows to bypass Slow Start and reach bottleneck capacity in the first RTT. Additionally, Petrov and Janevski [26] introduced **Advanced 5G-TCP**, which targets 5G backhaul by replacing linear window growth with a parabola-based response function to scale throughput rapidly without compromising fairness.

Advanced machine learning and heuristic models have been proposed to manage complex network dynamics. Khan et al. [14] and Alnawayseh et al. [2] explored **Hybrid Deep Learning** models using LSTM and SVM architectures to manage network slices. These models reside in the management layer and predict slice suitability and resource utilization to forestall slice failure. Malini and Karthigaikumar [19] introduced a **Weighted AIMD** algorithm (**WACC-D2DC**) using Hunger Games Search Optimization to dynamically tune congestion window weights based on real-time feedback. Similarly, Varala et al. [34] proposed **TCP DCERL+** for random loss environments, using an estimated bottleneck queue length to differentiate between wireless loss and congestion. Finally, Pazhooheshy et al. [25] proposed **Reminis**, which blends non-deterministic exploration with proactive measures. It uses a periodic "Guardian" module to categorize network conditions and combines user-space logic with kernel-space AIMD to maintain low delays.

Despite their variety, these algorithmic techniques necessitate intrusive host-side modifications and rely solely on indirect signals to infer the state of the RAN. This dependency on estimation rather than deterministic feedback leads to persistent inaccuracies in high-volatility 5G environments, where random wireless errors are frequently misinterpreted as congestion. Consequently, these endpoint-centric approaches remain inherently sub-optimal, as they lack the real-time visibility required to achieve absolute throughput and latency perfection.

2.2 RAN Optimizations

Complementing endpoint-centric improvements, the emergence of the Open RAN architecture has enabled researchers to implement intelligent control and resource optimization directly within the radio access network protocol stack. Nguyen et al. [23] and Kavehmadavani et al. [13] developed frameworks for **Intelligent Traffic Steering in 6G O-RAN**. The basic idea is a Joint Flow-split distribution, Congestion control, and Scheduling (JFCS) framework. These implementations utilize Multi-Agent Deep Reinforcement Learning (MADRL) within the Non-Real-Time RIC to manage frame-level resource block assignments. This logic allows for the coexistence of eMBB and uRLLC services by optimizing flow splitting at the RIC while managing power control at the Distributed Unit (DU). Complementing this, Lacava et al. [17] developed an **Intelligent Traffic Steering xApp** that maximizes throughput utility by making user-specific handover decisions based on Conservative Q-Learning (CQL) and user-level Key Performance Measurements (KPMs).

Network-wide awareness is also a critical component of RAN optimization. Arno Troch et al. [33] focused on **Transport Network (TN) Awareness**, where the RIC reallocates radio resources based on packet loss detected in the backhaul network, demonstrating the value of cross-domain telemetry. In terms of granular flow control, Irazabal et al. [11] proposed **TC-RAN**, a programmable system that treats individual data flows as first-class citizens. The implementation consists of a six-stage pipeline (Classifier, Policer, Queue, Scheduler, Shaper, and Pacer) managed by an O-RAN Service Model (E2SM-TC). This architecture is located between the SDAP and PDCP sublayers, allowing xApps to reconfigure fair queuing and pacing directly in the RAN.

Link-layer protocol enhancements are vital for mitigating bottlenecks. Srihari Das Gopinath et al. [9] addressed head-of-line blocking with **Enhanced PDCP Reordering**, which segregates flows at the PDCP layer and manages independent reordering timers for each. This implementation prevents packet loss in one flow from delaying healthy flows within the same Data Radio

Bearer (DRB). At the RLC layer, Mauricio et al. [20] provided an analysis of **RLC modes**, proving that Unacknowledged Mode (UM) can boost throughput by 20% in poor signal conditions by avoiding redundant retransmissions. These link-layer optimizations provide the underlying stability required for transport-layer optimization.

Finally, proactive cross-layer awareness has been explored for domain-specific tasks. Swaraj Kumar et al. [16] introduced a **Cognitive RAN-Aware** layer for NR-V2X, using a Sparse Recurrent Neural Network (SRNN) to predict RAN buffer bloat and trigger Explicit Congestion Notification (ECN) at the UPF. Similarly, Jacky Cao et al. [6] optimized transport for **Mobile Augmented Reality (MAR)** by demonstrating that increasing protocol buffer sizes on the 5G edge significantly reduces jitter for media streaming. To handle mmWave blockages, Takahashi and Nakao [32] proposed a **Cross-Layer Architecture** that classifies NLoS events into "weak" and "strong" categories, using SINR feedback from the physical layer to suppress window resets during outages. Lastly, Krasilov et al. [15] analyzed the **xStream platform**, which enables servers to set large initial windows based on link capacity estimates shared via an external signaling platform between network devices and the server.

While these RAN-integrated optimizations provide superior network visibility, they are constrained by significant architectural limitations. RIC-based systems often suffer from high control-loop latency, which can render optimization decisions obsolete by the time they are executed in millisecond-scale 5G environments. Furthermore, these platforms are typically restricted to a fixed set of exported Key Performance Indicators (KPIs), limiting their ability to adapt to specific or unforeseen transport-layer requirements. Finally, because these controllers operate off the data path, they remain incapable of making direct, on-the-fly modifications to user packets, relying instead on high-level signaling that lacks the precision required for real-time packet-level coordination.

2.3 L4S

Early L4S research in 5G focused on integrating dual-queue ECN marking and scalable congestion control within the New Radio (NR) protocol stack to mitigate the "visibility gap" caused by encrypted link-layer queues. Brunello et al. [5] implemented a marking strategy at the Packet Data Convergence Protocol (PDCP) layer of the gNB, where ECN bits are flipped based on real-time Radio Link Control (RLC) queue sojourn times. By pairing this RAN-side marking with the SCReAM (Self-Clocked Rate Adaptation for Multimedia) congestion controller at the application level, they demonstrated that latency-sensitive flows could maintain high throughput while keeping queuing delays significantly lower than traditional best-effort traffic. Similarly, Son et al. [30] took an end-to-end approach by integrating ECN-based L4S logic into the WebRTC framework. Their implementation utilizes an FQ-CoDel (Flow Queue Controlled Delay) scheduler in the base station to isolate L4S traffic, allowing the sender to adjust its rate precisely to avoid the latency spikes inherent in standard Google Congestion Control (GCC).

As the technology moved toward standardization, research shifted toward architectural compliance with 3GPP and O-RAN frameworks. Pan et al. [24] aligned L4S mechanisms with the 5G-Advanced Release 18 standard by mapping L4S flows to specific 5G QoS Identifiers (5QIs). Their work prototypes an L4S-GCC algorithm that performs ECN marking across multiple network nodes—including the User Plane Function (UPF) and the gNB—to achieve joint source-channel optimization. This standard-aligned approach allows the 5G network to securely expose congestion information to applications for rapid bitrate adaptation. Addressing the complexities of disaggregated architectures, Wan and Jamieson [35] proposed L4Span. This system resides in the CU-UP (Centralized Unit - User Plane) and utilizes F1-U interface feedback from the RLC layer in the DU (Distributed Unit) to predict future queue occupancy. A novel component of their implementation is a "feedback shortcircuiting" mechanism that modifies uplink TCP ACKs

in real-time, effectively bypassing 5G scheduling jitter and delivering immediate congestion signals to the content server.

Despite these advancements, standard 5G L4S implementations face critical barriers. First, the physical separation of the CU (where marking occurs) and the DU (where congestion happens) leads to *information aging*, where signals reaching the sender often reflect an obsolete channel state. Second, wireless links are too volatile for the static marking thresholds used in wired networks; a threshold optimized for a stable link often causes throughput collapse or excessive latency during mobility-induced fading. Finally, most current solutions are *intrusively hard-coded* into the RAN source code, making them vendor-dependent and incapable of adapting to specific application requirements without a fully programmable, inline framework.

2.4 Programmable Observability and User-Plane Telemetry

This section discusses the foundational research in packet-centric observability and programmable RAN telemetry that serves as the baseline and structural inspiration for our work.

The **Scout** telemetry framework, proposed by Seshasayee et al.[29], establishes the baseline for our approach to user-plane observability. The basic idea of Scout is to move away from aggregate, component-level Key Performance Indicators (KPIs) toward a packet-centric view that follows individual packet trajectories through the disaggregated RAN stack. The implementation involves inserting timestamped event markers at key touchpoints across the SDAP, PDCP, RLC, and MAC layers to capture a minimal set of metadata. This metadata is then processed through an *offline processing pipeline* to reconstruct packet trajectories and perform full-stack latency attribution, decomposing end-to-end delay into processing, queuing, and scheduling components. By tracing how transport-layer signatures—such as the periodic window cycles of TCP CUBIC—manifest within internal RAN buffers, Scout demonstrates that application-level performance can be directly explained by internal network dynamics. Our work adopts this trajectory-level analysis to provide a granular understanding of transport flows.

To implement granular observability programmatically and in real-time, we build upon the **Janus** framework introduced by Foukas et al. [8] The core idea of Janus is to provide a safe and flexible system for loading arbitrary monitoring and control "codelets" inline within RAN functions at runtime. While the original proposal discussed eBPF-based sandboxing, the framework utilizes a *user-space eBPF implementation* (ubpf) to enable high-performance telemetry collection and control without the security risks or overhead of kernel modifications. Janus provides programmable "hooks" at critical standard interfaces, such as FAPI and RRC, allowing for the collection of user-defined metrics and real-time decision-making at a millisecond scale. Unlike traditional RIC platforms that export a fixed set of KPIs, Janus enables arbitrary logic to be executed directly in the data path. By integrating these programmable hooks, Janus facilitates immediate interaction with the RAN state, providing the necessary infrastructure for the real-time aspects of our proposed optimization framework.

2.5 Synthesis and Research Gap

The body of research presented in this chapter highlights a fundamental tension between endpoint-centric protocol refinements and network-integrated architectural enhancements. While algorithmic modifications to protocols like NewReno and BBR provide robust mathematical frameworks for congestion control, they remain limited by their reliance on host-side modifications and indirect inference. Because these techniques must estimate the state of the RAN from end-to-end signals, they are prone to inaccuracies, often failing to distinguish between random wireless noise and true congestion.

Conversely, while optimizations within the RAN protocol stack provide the direct visibility that endpoint-centric approaches lack, they face significant systemic barriers. Current RIC-based architectures are constrained by high control-loop latency and a reliance on a fixed set of exported KPIs, which often renders optimization decisions obsolete in the face of millisecond-scale mmWave fluctuations. Furthermore, because these controllers typically operate off the data path, they lack the precision to perform direct, packet-level modifications required for real-time coordination. Standard L4S implementations in 5G further suffer from information aging and vendor-specific rigidity, hindering the deployment of application-adaptive marking policies.

These observations identify a critical research gap: the lack of a unified, high-fidelity framework that can bridge the divide between physical-layer KPMs and transport-layer pacing. RATO addresses this by synthesizing the programmable, user-plane observability of the Janus framework with deterministic, RAN-assisted control logic inspired by the trajectory-level analysis of Scout. By moving beyond high-latency RIC signaling toward a packet-centric, real-time optimization engine, RATO provides the necessary infrastructure to deliver an optimized end-to-end transport experience that is truly responsive to the dynamic environment of 5G and beyond networks.

3 Design

The goal of RATO is to enable transport-layer optimization through real-time telemetry and control within the Radio Access Network (RAN). Achieving this requires three key capabilities: (i) visibility into internal RAN state, (ii) low-overhead processing of telemetry data, and (iii) the ability to perform timely control actions that can influence transport-layer behavior.

To realize these capabilities, RATO is built on top of Janus [8], a programmable telemetry and control framework for network functions. Janus enables dynamic deployment of monitoring and control logic within the RAN, real-time telemetry collection, and packet-level actions directly on the datapath. Combined with the open-source srsRAN [31] platform, it provides a flexible foundation for implementing and evaluating transport-aware RAN functionality.

This chapter first presents the Janus architecture and its integration with srsRAN, followed by the experimental testbed used throughout the remainder of this work.

3.1 Janus

Janus [8] is a programmable telemetry and control framework that enables safe, low-overhead execution of eBPF codelets within user-space network functions. In this work, we use a jBPF-instrumented version of srsRAN [7], which provides programmable hookpoints throughout the RAN stack and supports dynamic deployment of codelets at runtime. This allows RATO to observe internal RAN state, collect telemetry, and perform packet-level control actions directly in the datapath without invasive modifications to the RAN implementation.

The Janus architecture (Fig. 3.1) consists of the following components:

- **jBPF [21]:** A user-space eBPF instrumentation framework that dynamically attaches codelets to predefined hookpoints using dynamic binary instrumentation. This preserves the semantics of eBPF hooks while avoiding kernel traps and context switches.
- **Verifier and uBPF Runtime [28]:** A user-space verifier enforces memory safety, bounded loops, and register correctness, providing guarantees similar to those of the in-kernel eBPF verifier. Verified codelets are executed by uBPF, a lightweight virtual machine embedded within the RAN process.
- **Shared eBPF Maps:** Shared maps provide persistent state across hookpoints and codelet invocations, enabling state sharing and coordination between codelets.
- **JRT Controller [22]:** A centralized controller responsible for managing applications and codelets, allowing them to be loaded, updated, or removed without restarting the RAN. Applications can subscribe to telemetry streams generated by codelets and send control messages in return. This communication is exposed through the controller's router component, which provides a stream-based I/O API between applications and codelets.

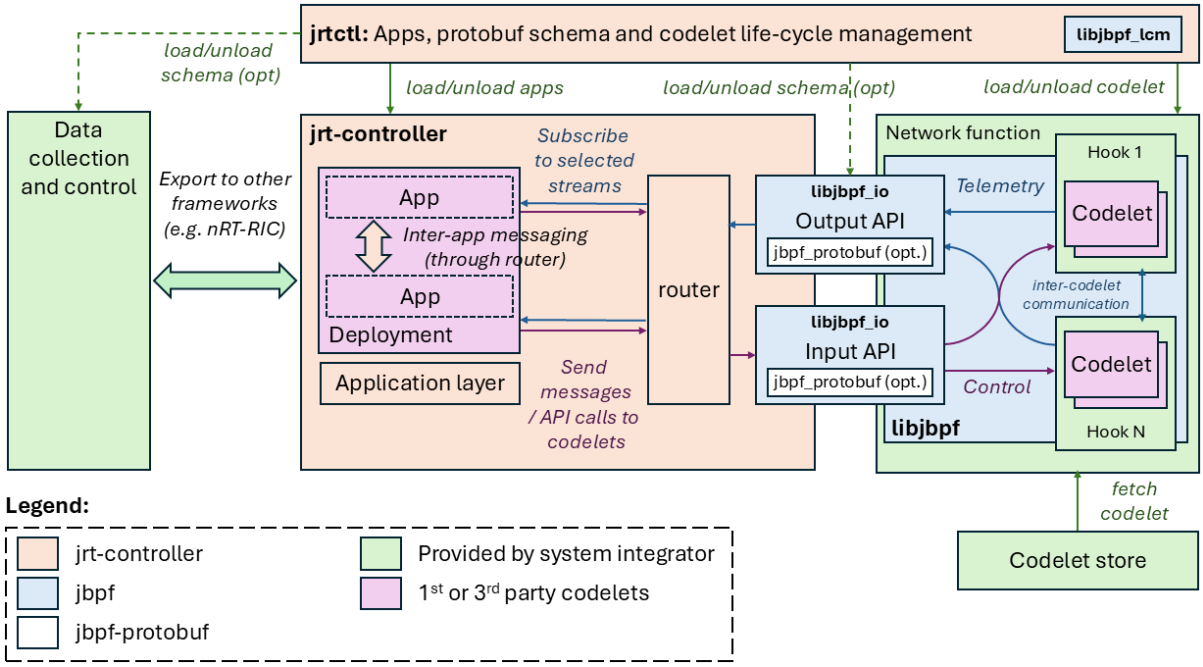

Figure 3.1: Janus Architecture

- **Shared-Memory I/O Channels:** The jBPF runtime and JRT Controller communicate through shared-memory channels, enabling zero-copy transfer of telemetry and control messages. Applications access these channels through the router API, while codelets emit and consume messages through lock-free ring buffers, ensuring low-latency communication without blocking the RAN datapath.

In addition to the hookpoints already provided by the jBPF-instrumented srsRAN stack, we introduce several custom hookpoints to expose the RAN state required by RATO. The flexibility of the instrumentation framework allows these hookpoints to be added with minimal changes to the underlying RAN implementation. We further develop a collection of eBPF codelets and controller applications that leverage both existing and custom hookpoints to implement RATO's telemetry and control mechanisms.

To improve observability, we also implement all logging points used by Scout [29] as Janus hookpoints. This exposes Scout's telemetry as real-time event streams, eliminating the need for traditional logging and post-processing pipelines.

3.2 Testbed

We deploy a complete end-to-end 5G testbed (Figure 3.2) integrated with Janus and the components required for telemetry collection, control, and evaluation. All core components are deployed as Docker containers to simplify management and ensure reproducibility.

The testbed consists of the following components:

- **Open5GS**: Provides the 5G Core Network, including the user-plane and control-plane functions required for UE connectivity.
- **srsRAN gNB**: A 5G gNB instrumented with the Janus jBPF library, enabling the deployment of eBPF codelets and access to programmable hookpoints within the RAN stack.
- **JRT Controller**: The Janus runtime controller responsible for managing applications, codelets, telemetry streams, and control messages.

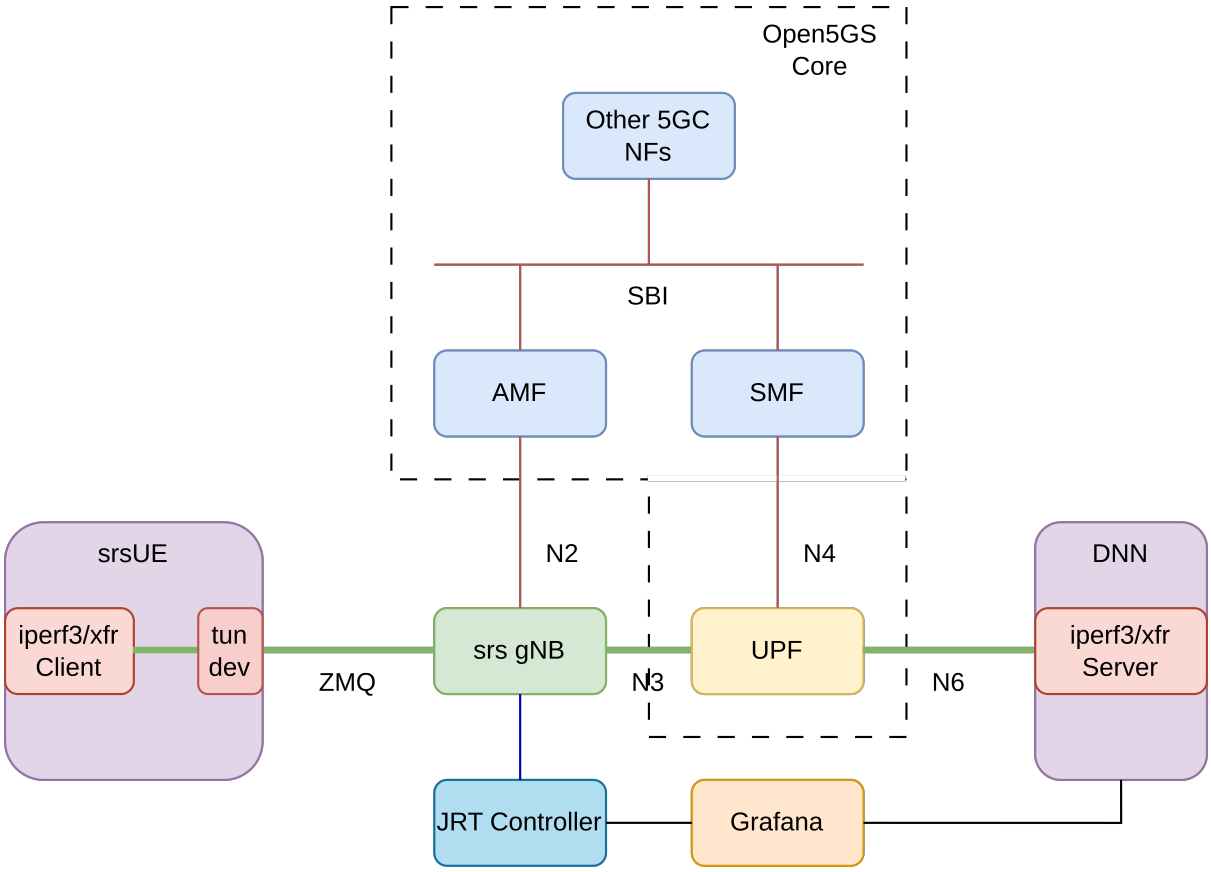

Figure 3.2: Experimental Testbed

- **srsUE**: A software UE that connects to the gNB over a ZMQ-based radio interface. Each UE creates a TUN interface through which all user traffic is routed into the 5G network.
- **GNU Radio**: Used in multi-UE experiments to multiplex and demultiplex ZMQ streams between multiple srsUE instances and the gNB.
- **Grafana**: Provides real-time visualization of both RAN telemetry and transport-layer performance metrics.

A separate host acts as the Data Network (DN) server and is used as the source and sink of application traffic. This separation ensures that traffic traverses the complete 5G stack, including the RAN and Core Network, before reaching the application endpoint.

For traffic generation and performance evaluation, we use two speed-test and load-generation tools: *iperf3* and *xfr*. *iperf3* is a widely used network benchmarking tool, while *xfr* is a newer speed-test framework that provides additional measurement and workload-generation capabilities. Multiple *iperf3* and *xfr* server instances are hosted on the DN server to support both upload and download experiments.

4 Transport State Estimation

Although RATO is designed as a general framework for transport-layer optimization, our initial implementation focuses exclusively on TCP due to its widespread deployment and well-understood congestion control behavior. The same approach can be extended to other transport protocols, such as QUIC and SCTP, in future work.

To simplify the initial design and evaluation, we make two assumptions. First, each UE is assumed to carry at most one active TCP flow at a time. This avoids the need to maintain per-flow state within the RAN, although support for multiple concurrent flows could be added by tracking additional packet header fields, such as the flow 5-tuple. Second, we focus exclusively on downlink traffic, as the downlink path is typically the primary bottleneck in cellular networks. Neither assumption is fundamental to RATO and both can be relaxed in future implementations.

We focus on three key TCP performance indicators: (i) throughput, (ii) round-trip time (RTT), and (iii) congestion window (CWND). The latter two strongly influence congestion control behavior and, consequently, the achievable throughput. CWND also closely correlates with the number of bytes in flight, i.e., data that has been transmitted but not yet acknowledged.

Scout [29] identifies the RLC queue as one of the most significant bottlenecks affecting TCP performance on the downlink path. Located at the edge of the RAN, the RLC queue is the final buffering point before packets are transmitted over the air interface. Excessive queue occupancy can increase latency, reduce throughput, and ultimately lead to packet loss.

Motivated by these observations, RATO tracks three RLC queue metrics: (i) dequeue rate, (ii) queueing delay, and (iii) queue occupancy. These metrics are expected to correlate with the corresponding TCP performance indicators of throughput, RTT, and inflight bytes, respectively. By monitoring these RAN-level signals, RATO aims to infer transport-layer behavior directly within the RAN and enable timely optimization decisions.

4.1 In-RAN Monitoring

To track RLC queue dynamics, we introduce two custom hookpoints into the srsRAN downlink path:

- **rlc_dl_internal_enq**: Triggered when a new SDU is delivered from PDCP to RLC. It exposes the UE index, DRB ID, SDU length, and PDCP sequence number.
- **rlc_dl_internal_deq**: Triggered when an RLC PDU is dequeued for transmission. It exposes the same metadata as the enqueue hookpoint, allowing enqueue and dequeue events to be correlated.

In addition, we leverage a pre-existing maintenance hookpoint, *periodic_call*, which executes every 10 ms in a dedicated maintenance thread outside the datapath. Since telemetry is reported in 100 ms intervals, only every tenth invocation is used for metric aggregation and reporting.

Algorithm 1 `rlc_dl_internal_enq` codelet

$ts \leftarrow$ current timestamp
Extract UE index, PDCP sequence number, and SDU length
$snKey \leftarrow$ (UE index, PDCP SN)
Store ts in `EnqueueTimestampMap[snKey]`
$bucketId \leftarrow \lfloor ts / BucketDuration \rfloor$
$bucketKey \leftarrow$ (UE index, $bucketId$)
Increment `EnqueueBytes[bucketKey]` by SDU length

Algorithm 2 `rlc_dl_internal_deq` codelet

$ts \leftarrow$ current timestamp
Extract UE index, PDCP sequence number, and SDU length
if flow is not a DRB operating in AM mode **then**
    return
end if
$snKey \leftarrow$ (UE index, PDCP SN)
$enqTs \leftarrow$ `EnqueueTimestampMap[snKey]`
$queueDelay \leftarrow ts - enqTs$
Emit queue-delay telemetry event
$bucketId \leftarrow \lfloor ts / BucketDuration \rfloor$
$bucketKey \leftarrow$ (UE index, $bucketId$)
Increment `DequeueBytes[bucketKey]` by SDU length

RATO uses three eBPF codelets to collect and process RLC telemetry:

- **rlc_dl_internal_enq** (Algorithm 1): Records the enqueue timestamp of each SDU and updates a per-UE counter of enqueued bytes.
- **rlc_dl_internal_deq** (Algorithm 2): Retrieves the corresponding enqueue timestamp, computes the queueing delay, and updates a per-UE counter of dequeued bytes.
- **rlc_dl_periodic** (Algorithm 3): Executes from the maintenance thread to compute queue occupancy from cumulative enqueue and dequeue byte counts, emit telemetry events, and reset counters for the next reporting interval.

Using these codelets, RATO exports the following RLC metrics:

1. **RLC Dequeue Rate**: Total dequeued bytes divided by the reporting interval duration.
2. **RLC Queueing Delay**: Time elapsed between the enqueue and dequeue events of an SDU.
3. **RLC Queue Occupancy**: Number of bytes currently buffered in the RLC queue, computed from cumulative enqueue and dequeue byte counts.

Enqueue and dequeue byte counters are maintained in fixed-duration 100 ms buckets. This

Algorithm 3 `rlc_dl_periodic` codelet

$bucketId \leftarrow \lfloor currentTime / BucketDuration \rfloor$
for all UEs **do**
    Read `EnqueueBytes[UE, bucketId-1]`
    Read `DequeueBytes[UE, bucketId-1]`
    $Q \leftarrow Q + EnqueueBytes - DequeueBytes$
    Emit queue-occupancy telemetry event
    Reset enqueue/dequeue counters for the completed bucket
end for

serves two purposes: it enables computation of dequeue rates over a well-defined observation window and significantly reduces telemetry overhead. Emitting telemetry on a per-packet basis would generate a high-frequency event stream that could overwhelm the controller and increase runtime overhead. Aggregating metrics over short intervals provides timely visibility into RLC behavior while maintaining efficient operation.

Finally, a controller application subscribes to the emitted telemetry streams and exports them directly to Grafana for real-time visualization. This enables continuous monitoring of RLC queue dynamics and facilitates correlation with TCP performance indicators measured at the endpoints.

4.2 Ground Truth Validation

To validate the relationship between RAN telemetry and transport-layer behavior, we require accurate measurements of TCP performance indicators such as throughput, RTT, and congestion window (CWND). For this purpose, we develop *TcpCap*, a packet-capture-based monitoring system that observes TCP traffic directly on the server and reconstructs the TCP state machine from captured packets.

TcpCap is implemented in Rust using libpcap and operates by passively monitoring all TCP packets transmitted and received by the server. For each connection, it tracks sequence numbers, acknowledgements, retransmissions, and timing information to reconstruct the sender's view of the TCP connection. Using this reconstructed state, TcpCap derives several performance indicators, including throughput, RTT, smoothed RTT (sRTT), and bytes-in-flight.

Metrics are computed on every ACK, aggregated into 100 ms buckets, and exported to Grafana for visualization and correlation with RAN telemetry. By observing every packet in both directions, TcpCap provides a detailed and implementation-independent view of TCP behavior, making it well suited for establishing ground truth against which RATO's RAN-derived measurements can be evaluated.

TcpWire

TcpWire is a lightweight Rust-based monitoring tool that uses an eBPF program of type `BPF_PROG_TYPE_SOCK_OPS` to monitor TCP flow metrics directly within the Linux kernel. The program enables TCP RTT callbacks and extracts metrics such as sRTT, CWND, and acknowledged bytes from the TCP stack. These events are delivered to a user-space daemon through a ring buffer, aggregated into 100 ms buckets, and exported to Grafana for visualization.

Compared to TcpCap, TcpWire incurs significantly lower overhead and provides direct access to the kernel's internal TCP state. However, it is tied to Linux TCP internals and only exposes information made available through SOCK_OPS callbacks. Its event rate is also determined by kernel callback behavior rather than individual packet events, making it less suitable as a comprehensive ground-truth source. Nevertheless, TcpWire serves as a useful low-overhead validation tool by exposing TCP state directly from the kernel, providing a perspective that closely reflects the sender's internal view of the connection.

Figure 4.1: Grafana dashboard showing real-time RLC and TCP telemetry collected by RATO and TcpCap.

4.3 Results

Figure 4.1 shows the Grafana dashboard used to visualize both the RLC telemetry exported by RATO and the TCP performance indicators collected by TcpCap. The dashboard provides a real-time view of throughput, RTT, bytes in flight, dequeue rate, queueing delay and queue occupancy.

The collected telemetry demonstrates a strong correlation between the TCP and RLC metrics. In particular, variations in TCP throughput closely track changes in the RLC dequeue rate, TCP RTT follows trends in the measured RLC queueing delay, and changes in the TCP congestion window are reflected in the observed RLC queue occupancy. These observations support the hypothesis that RLC queue dynamics provide a useful proxy for transport-layer behavior.

Furthermore, the results demonstrate that the proposed telemetry pipeline is capable of exporting these metrics in real time. By combining in-RAN telemetry with endpoint measurements on a common Grafana dashboard, RATO enables direct observation of the relationship between transport-layer performance and RAN behavior, laying the foundation for the transport optimization mechanisms presented in the following chapter.

5 RAN-Assisted ECN and L4S

The telemetry mechanisms described in the previous chapter enable RATO to observe queue dynamics throughout the RAN data path. The next step is to use this information to influence transport-layer behavior. As an initial proof of concept, we implement a RAN-assisted ECN marking scheme inspired by L4Span [35], allowing congestion information observed within the RAN to be communicated directly to transport endpoints.

Explicit Congestion Notification (ECN) provides a standardized mechanism for signaling congestion without dropping packets. Instead of relying on packet loss, network elements can mark packets with the Congestion Experienced (CE) codepoint, allowing transport protocols to react to congestion before losses occur. L4S extends ECN by using marking as a continuous congestion signal. Rather than waiting for severe congestion, packets are marked as queue occupancy begins to increase, enabling transport protocols to make small and frequent rate adjustments while maintaining low queueing delay.

L4Span adapts this idea to cellular networks by performing ECN marking within the RAN. Building on this approach, RATO uses its real-time RAN telemetry to estimate queue state and generate congestion signals that closely reflect current radio access conditions.

5.1 Design

Implementing L4S-style marking requires two capabilities: (i) tracking packets throughout the RAN stack to estimate queue state, and (ii) using this state to generate ECN marking decisions. To support these functions, RATO introduces a new hookpoint, *pdcp_dl_sdu_segment*, and reuses the existing *rlc_dl_sdu_send_completed* hookpoint.

The PDCP sequence number serves as a packet identifier throughout the PDCP and RLC layers. RATO therefore uses the tuple *(UE, DRB, PDCP SN)* to correlate events across layers and maintain per-packet state.

Queue State Estimation

To estimate queue state, RATO tracks packets from the point at which a PDCP sequence number is assigned until the corresponding packet is delivered to the MAC layer. The PDCP sequence number provides a stable identifier that remains available throughout PDCP and RLC processing, allowing enqueue and dequeue events to be correlated using the tuple *(UE, DRB, SN)*.

To support packet tracking, we introduce a new custom hookpoint, *pdcp_dl_sdu_segment*, specifically for RATO. The hookpoint is triggered immediately after a PDCP sequence number is assigned to an SDU and exposes metadata including the UE index, DRB identifier, and PDCP sequence number. This allows RATO to initialize per-packet state and track packets consistently across the PDCP and RLC layers.

Algorithm 4 Queue-State Estimation

procedure PACKETENQUEUE($pktSize$)
    $Q \leftarrow Q + pktSize$
end procedure
procedure PACKETDELIVERY($pktSize, ts$)
    $Q \leftarrow \max(0, Q - pktSize)$
    if $lastTs \neq 0$ **then**
        $R_{inst} \leftarrow \dfrac{pktSize}{ts - lastTs}$
        Update dequeue-rate estimate $\hat{R}$ using EWMA
        $E \leftarrow |R_{inst} - \hat{R}|$
        Update prediction-error estimate $\hat{E}$ using EWMA
    end if
    if $\hat{R} > 0$ **then**
        $\hat{d}_q \leftarrow Q/\hat{R}$
    else
        $\hat{d}_q \leftarrow 0$
    end if
    $lastTs \leftarrow ts$
end procedure

To observe packet departures, RATO reuses the existing *rlc_dl_sdu_send_completed* hookpoint, which is triggered when an RLC PDU has been successfully delivered to the MAC layer. Using the packet identifier established at PDCP, the corresponding packet state can be retrieved and removed.

Two eBPF codelets attached to these hookpoints maintain a per-DRB estimate of the standing queue size, dequeue rate, dequeue-rate prediction error, and queueing delay. Packet arrivals increase the standing queue estimate, while packet deliveries decrease it. Upon each delivery event, the codelet computes an instantaneous dequeue rate and updates exponentially weighted moving average (EWMA) estimates of both the dequeue rate and its prediction error. These quantities are subsequently used by the marking logic described in the next subsection.

The queue-state update procedure is summarized in Algorithm 4.

ECN Marking

RATO performs ECN marking directly within the PDCP processing path using a second codelet attached to the *pdcp_dl_sdu_segment* hookpoint. In addition to packet metadata, this hookpoint exposes pointers to the generated SDU segments, allowing the codelet to access packet buffers directly. As a result, packet headers can be inspected and modified in-line without requiring additional packet copies or changes to the underlying RAN processing logic.

For each IPv4 packet, the codelet determines whether the packet is ECN-capable and identifies its traffic class from the ECN field. Packets marked as `ECT(1)` are treated as L4S traffic, whereas packets marked as `ECT(0)` are treated as classic ECN traffic.

For L4S traffic, the codelet computes the service rate required to drain the current queue within a target queueing delay and compares it against the predicted dequeue rate obtained from the queue-state estimator. If the predicted service rate is sufficient, packets are not marked. As the required rate exceeds the predicted rate, the marking probability increases progressively from zero to one. The dequeue-rate prediction error is used to smooth the transition between these operating regions, reducing sensitivity to short-term fluctuations in radio conditions.

Algorithm 5 L4S Marking Probability

procedure COMPUTEMARKPROBABILITY($Q, \hat{R}, \hat{E}$)
  $R_{req} \leftarrow Q / D_{target}$
  $R_{low} \leftarrow \max(0, \hat{R} - \hat{E})$
  $R_{high} \leftarrow \hat{R} + \hat{E}$
  if $R_{req} \leq R_{low}$ **then**
    $p \leftarrow 0$
  else if $R_{req} \geq R_{high}$ **then**
    $p \leftarrow 1$
  else
    $p \leftarrow \dfrac{R_{req} - R_{low}}{R_{high} - R_{low}}$
  end if
  Mark packet with probability p
end procedure

Packets selected for marking have their ECN field updated to the Congestion Experienced (CE) codepoint directly within the packet buffer. Since the modification occurs within the RAN datapath, congestion information can be communicated to transport endpoints without requiring packet loss or changes to endpoint implementations.

The marking procedure is summarized in Algorithm 5.

5.2 Results

To evaluate the effectiveness of RATO's ECN/L4S marking mechanism, we conduct experiments under three scenarios: (i) a single UE, (ii) two UEs with ECN/L4S marking enabled for both UEs, and (iii) two UEs where ECN/L4S marking is enabled for only one UE. In each case, we compare TCP performance with and without the marking codelets loaded in the RAN.

The UE(s) continuously download data from the DNN server using the *xfr* [18] speed test application for a duration of five minutes, creating sustained downlink traffic and allowing the long-term effects of the marking mechanism to be observed. We measure throughput, RTT, congestion window (CWND), and retransmissions to evaluate the impact of RATO on transport-layer behavior.

5.2.1 Single UE

For the single-UE scenario, the experiment is repeated using three TCP congestion control algorithms: BBR, Cubic, and Reno.

Figure 5.1 summarizes the aggregate TCP performance metrics across all runs. The most significant impact is observed for Cubic, which supports ECN signaling. Figure 5.1(ii) shows that enabling ECN/L4S marking reduces the TCP RTT from approximately 400–500 ms to 40–50 ms, representing nearly an order-of-magnitude reduction in latency. A corresponding reduction is visible in the congestion window distribution shown in Figure 5.1(iii), indicating that Cubic maintains substantially fewer bytes in flight when congestion information is provided through ECN marks. Figure 5.1(iv) further shows that retransmissions are reduced to almost zero, suggesting that congestion is signaled early enough to prevent queue overflow and packet loss.

Despite the significant reduction in latency and congestion window size, Figure 5.1(i) shows that Cubic maintains a similar average throughput with and without marking. However, the throughput distribution is noticeably tighter when marking is enabled, indicating more stable

transport behavior and reduced throughput variability.

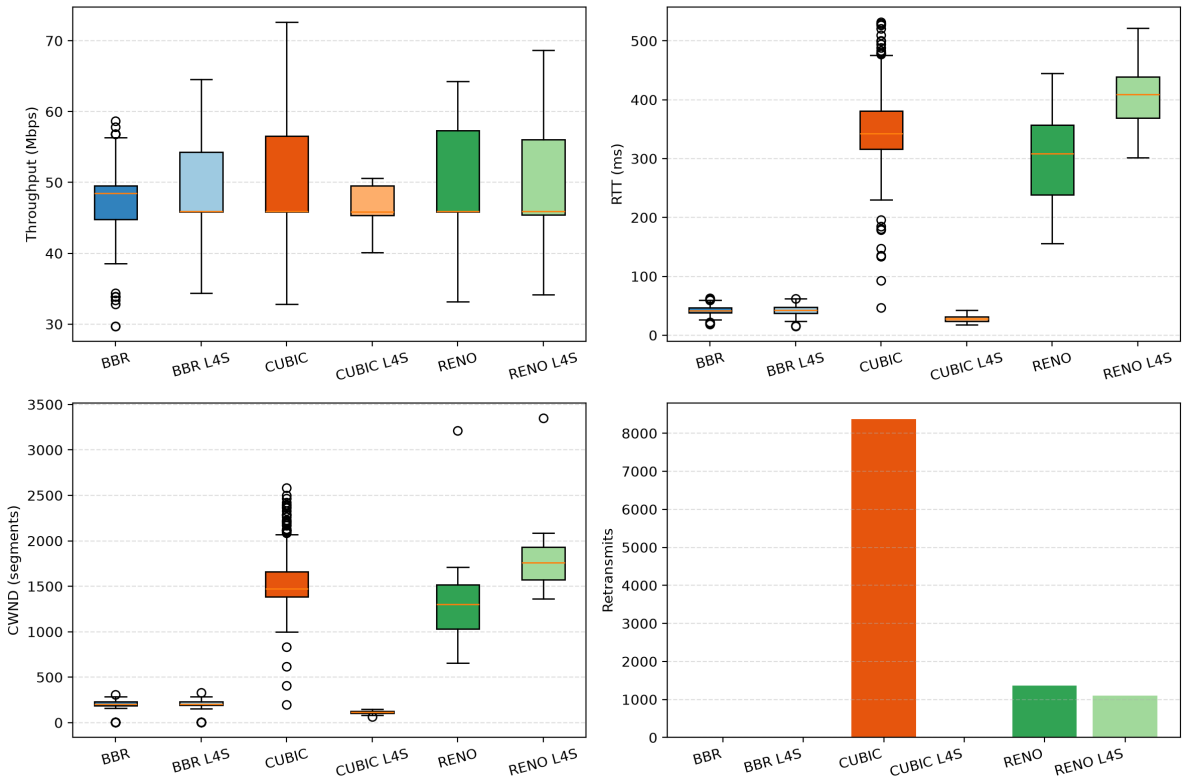

Figure 5.1: Aggregate TCP performance for a single UE with and without RATO ECN/L4S marking. ECN-aware Cubic achieves substantially lower RTT, smaller congestion windows, and fewer retransmissions while maintaining comparable throughput. Top-left (i): TCP Throughput, Top-right (ii): TCP RTT, Bottom-left (iii): TCP CWND, Bottom-right (iv): TCP Retransmissions

As expected, BBR and Reno show little to no change when ECN/L4S marking is enabled, since neither congestion control algorithm reacts to ECN marks in our setup. Although Figure 5.1(ii) shows a slightly higher RTT for Reno when marking is enabled, repeated experiments revealed no consistent trend. Across multiple runs, the RTT was observed to be comparable, higher, or even lower than the baseline case, indicating that the difference is unrelated to the marking mechanism and is instead attributable to normal experimental variability.

Figure 5.2 provides a time-series view of Cubic's behavior and confirms the aggregate observations. Throughout the duration of the experiment, RTT remains consistently around 40–50 ms when marking is enabled, compared to several hundred milliseconds without marking. Similarly, the congestion window remains significantly smaller while throughput remains comparable. The throughput trace is also noticeably smoother, exhibiting fewer fluctuations and a more stable operating point. Together, these results demonstrate that RATO's ECN/L4S marking mechanism can substantially reduce latency and queue occupancy while preserving throughput.

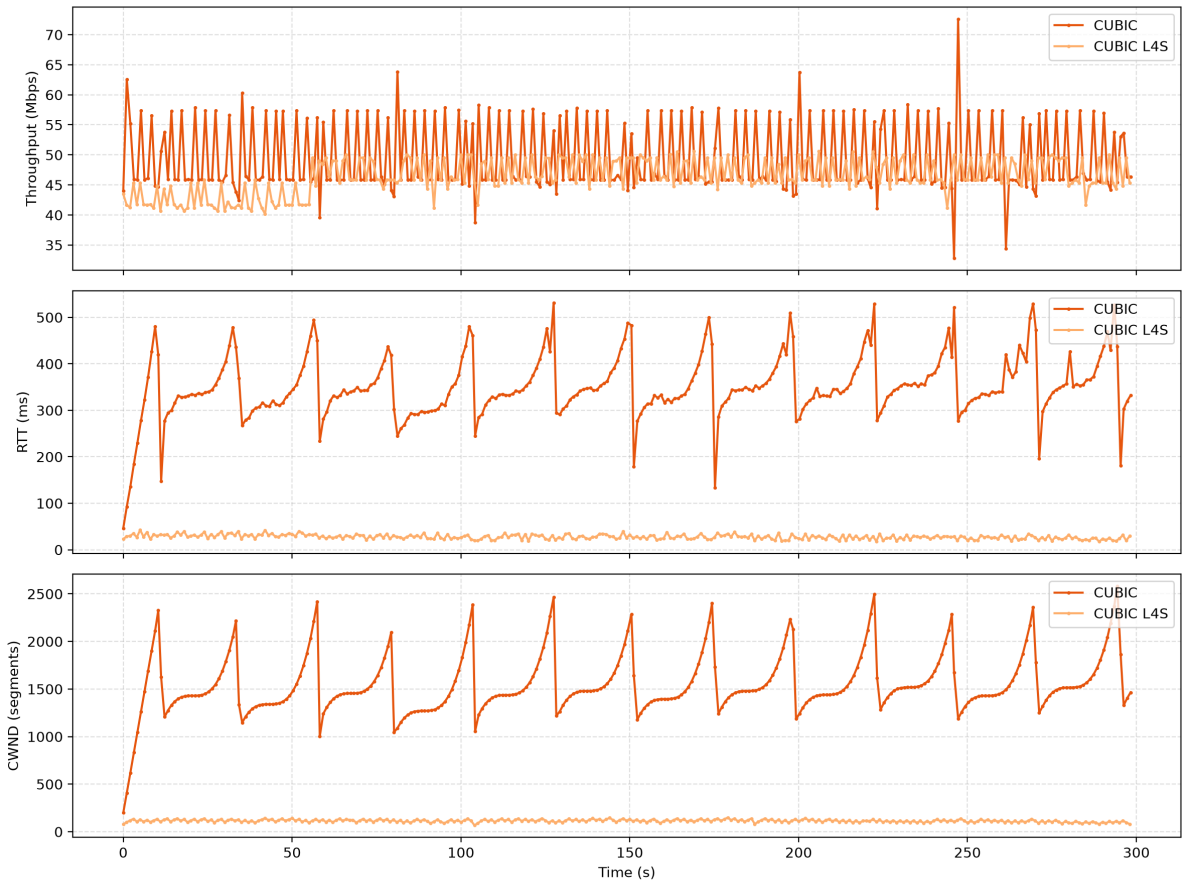

Figure 5.2: Time-series TCP performance for Cubic congestion control with and without RATO ECN/L4S marking in the single-UE scenario. ECN/L4S marking significantly reduces RTT and CWND while preserving throughput and improving stability throughout the experiment. Top to bottom: (i) TCP Throughput, (ii) TCP RTT, (iii) TCP CWND

5.2.2 Multiple UEs

To evaluate whether the proposed mechanism scales beyond a single flow, we repeat the experiment with two UEs simultaneously downloading data from the DNN server using Cubic congestion control. As in the single-UE experiment, we compare two configurations: one with the ECN/L4S marking codelets enabled in the RAN and one without them.

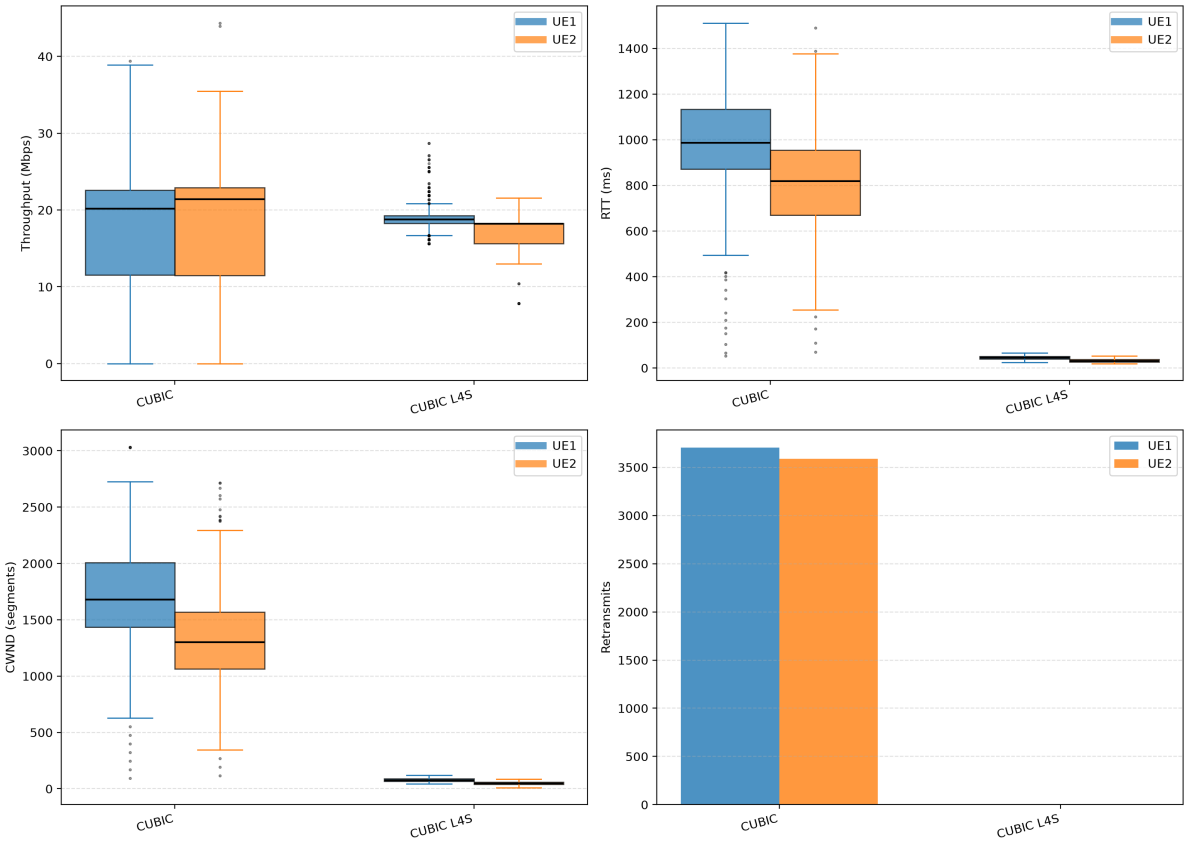

Figure 5.3: Aggregate TCP performance for two UEs with and without RATO ECN/L4S marking. Similar to the single-UE case, enabling ECN/L4S marking reduces RTT from approximately 1000–1400 ms to 40–50 ms while maintaining comparable throughput. Congestion window size and retransmissions are also significantly reduced for both UEs. Top-left (i): TCP Throughput, Top-right (ii): TCP RTT, Bottom-left (iii): TCP CWND, Bottom-right (iv): TCP Retransmissions.

The results shown in Figures 5.3 and 5.4 closely match the observations from the single-UE experiment. For both UEs, enabling ECN/L4S marking reduces RTT from approximately 1000–1400 ms to 40–50 ms, representing more than an order-of-magnitude reduction in latency. This reduction is accompanied by a substantially smaller congestion window and near-zero retransmissions, indicating that congestion is signaled before packet loss occurs. Despite the dramatic reduction in latency and queue occupancy, throughput remains comparable to the baseline case.

These results demonstrate that the proposed RAN-assisted marking mechanism continues to operate effectively in the presence of multiple concurrent UEs. The benefits observed for a single flow are preserved without introducing instability or unfairness, suggesting that the approach scales naturally as additional UEs share the RAN resources.

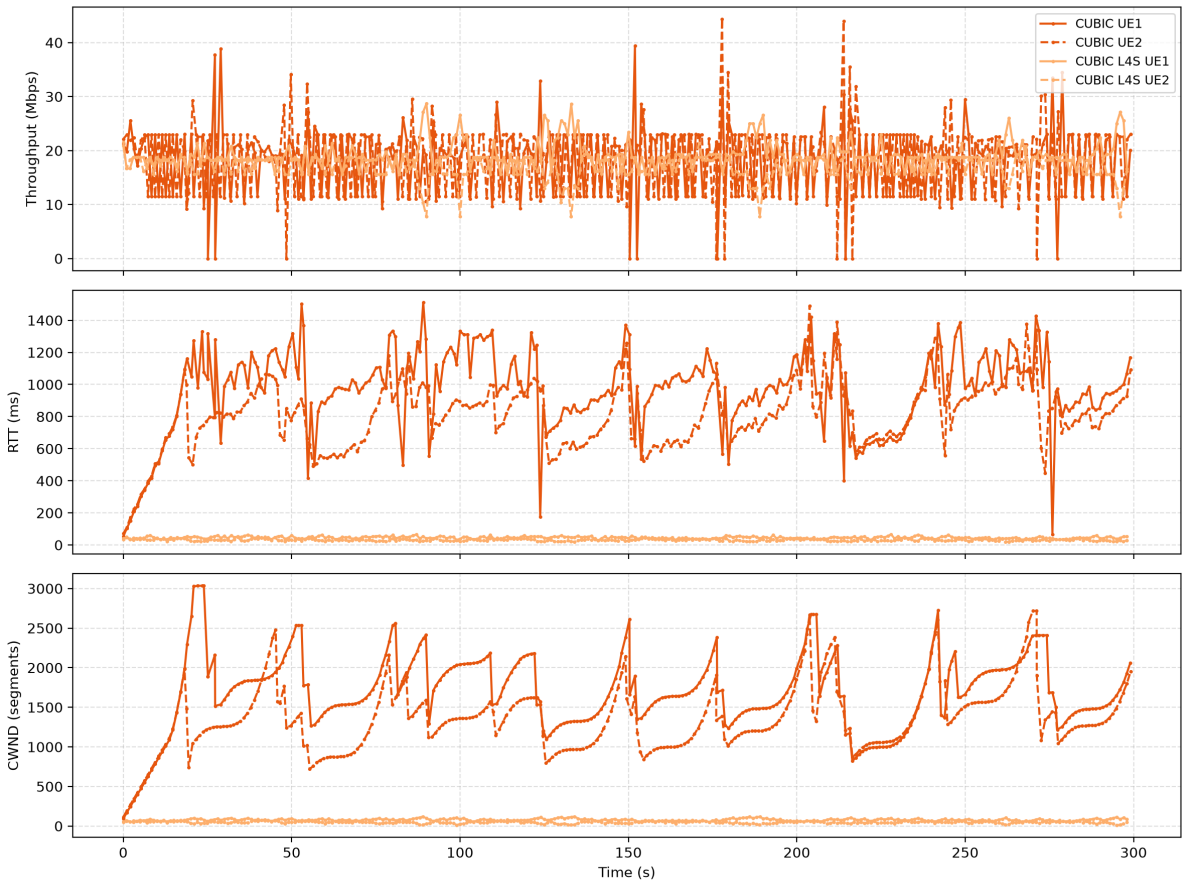

Figure 5.4: Time-series TCP performance for two UEs running Cubic congestion control with and without RATO ECN/L4S marking. For both UEs, ECN/L4S marking consistently maintains RTTs around 40–50 ms, compared to 1000–1400 ms without marking, while preserving throughput and reducing congestion window size throughout the experiment. Top to bottom: (i) TCP Throughput, (ii) TCP RTT, (iii) TCP CWND.

5.2.3 Mixed UEs

The mixed-UE experiment evaluates a heterogeneous deployment in which UE1 operates without ECN/L4S marking, while UE2 is configured to receive RATO's ECN/L4S congestion signals. Both UEs simultaneously download data from the DNN server using Cubic congestion control.

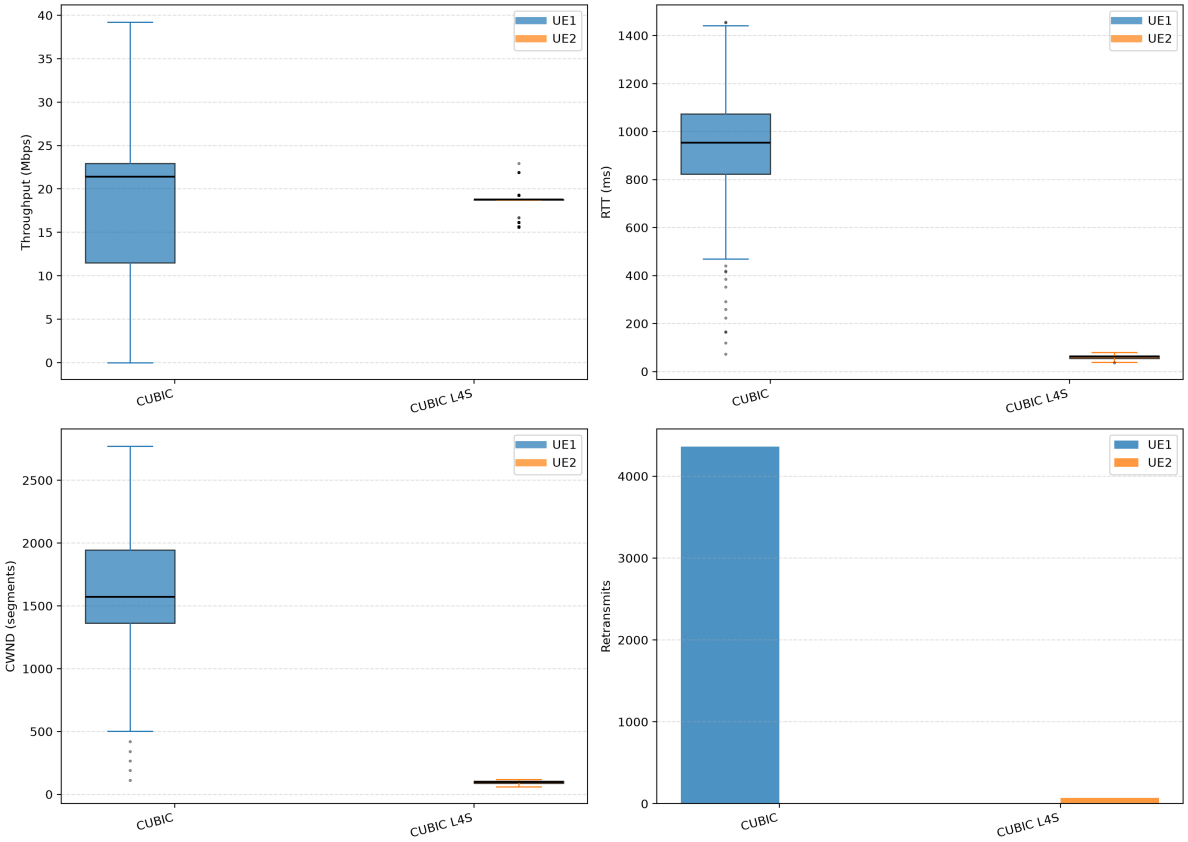

Figure 5.5: Aggregate TCP performance for a mixed-UE deployment where UE1 operates without ECN/L4S marking and UE2 receives RATO ECN/L4S marking. UE1 experiences RTTs of approximately 1000–1400 ms, while UE2 maintains RTTs of only 50–60 ms, demonstrating the effectiveness of selective per-flow marking. Throughput remains comparable for both UEs. Top-left (i): TCP Throughput, Top-right (ii): TCP RTT, Bottom-left (iii): TCP CWND, Bottom-right (iv): TCP Retransmissions.

The results (Figure 5.5 and 5.6) show a clear divergence in transport behavior between the two UEs. UE1 exhibits RTTs of approximately 1000–1400 ms, similar to the baseline experiments without marking. In contrast, UE2 maintains RTTs of only 50–60 ms throughout the experiment, achieving more than an order-of-magnitude reduction in latency. Despite this difference in queue occupancy and RTT, both UEs are able to sustain comparable throughput.

These results demonstrate that RATO's marking mechanism can be selectively applied on a per-flow basis. ECN-enabled flows benefit from substantially lower latency without requiring changes to other flows sharing the same RAN infrastructure, highlighting the incremental deployability of the approach.

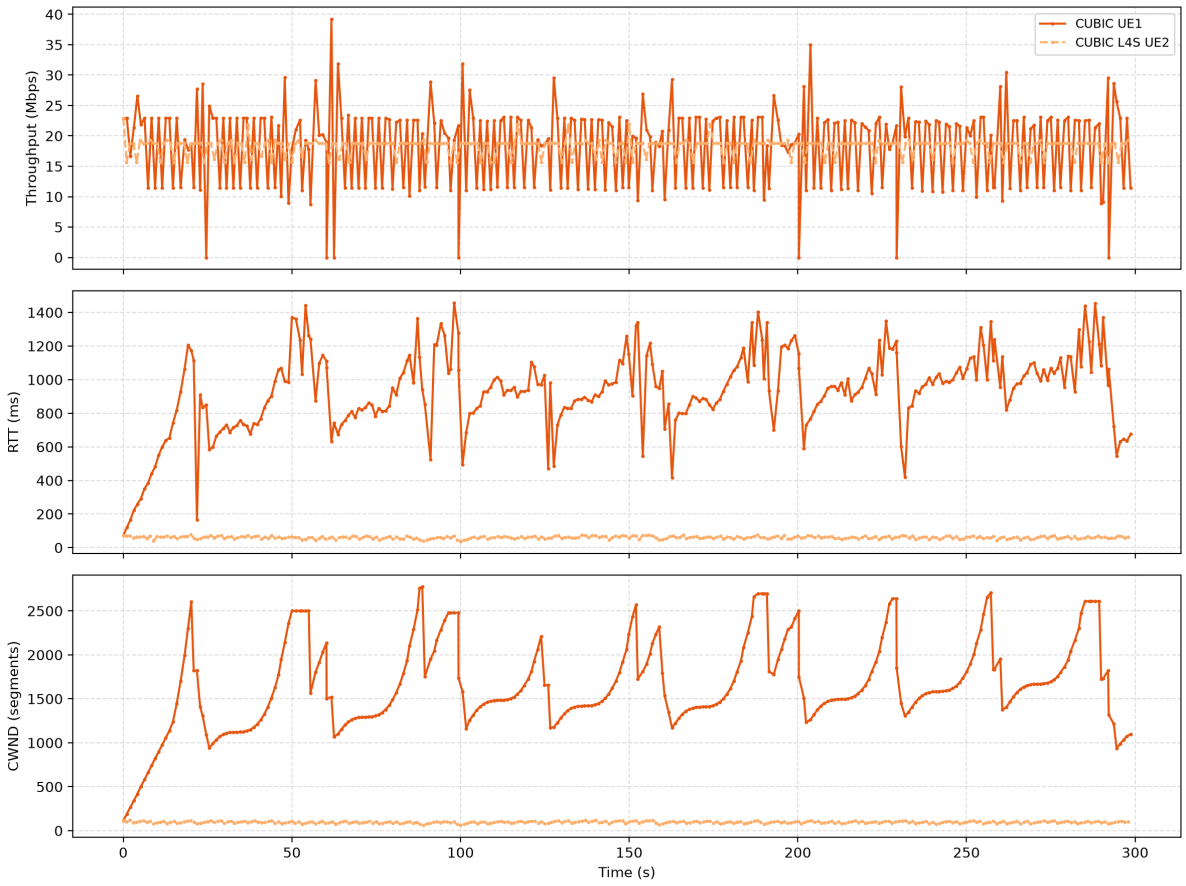

Figure 5.6: Time-series TCP performance for the mixed-UE deployment. While UE1 continues to operate with RTTs exceeding 1 s, UE2 maintains RTTs of approximately 50–60 ms due to RATO ECN/L4S marking. The results demonstrate that marked and unmarked flows can coexist within the same RAN while exhibiting distinct congestion-control behavior. Top to bottom: (i) TCP Throughput, (ii) TCP RTT, (iii) TCP CWND.

6 Conclusion

Transport protocols play a critical role in determining the end-to-end performance experienced by applications in 5G networks. However, modern congestion control algorithms operate using only end-to-end observations and have limited visibility into the rapidly changing conditions within the Radio Access Network (RAN). As a result, transport protocols may react suboptimally to wireless-induced performance variations, leading to increased latency, unstable throughput, and inefficient utilization of available radio resources.

To address this challenge, this work presented **RATO (RAN-Assisted Transport Optimization)**, a framework built on top of the Janus programmable networking platform that enables transport-aware telemetry and control within the 5G RAN. By leveraging programmable eBPF codelets deployed directly within the RAN, RATO exposes transport-relevant information such as queue occupancy, queueing delay, and dequeue rate in real time, while also providing mechanisms for implementing transport-aware control policies.

We demonstrated the feasibility of the approach through the implementation of a complete telemetry pipeline, transport-state monitoring infrastructure, and a proof-of-concept RAN-assisted ECN/L4S marking mechanism. Experimental evaluation on an end-to-end 5G testbed showed that the exported RAN telemetry closely correlates with observed transport behavior. Furthermore, the ECN/L4S marking mechanism significantly reduced latency for ECN-capable congestion control algorithms while maintaining comparable throughput, demonstrating the potential benefits of incorporating RAN knowledge into transport-layer decision making.

Overall, the results highlight the value of exposing internal RAN state to transport protocols and demonstrate that programmable RAN platforms can serve as an effective foundation for cross-layer transport optimization in future mobile networks.

6.1 Future Work

Several directions remain for extending the capabilities of RATO:

- **Per-UE, Per-Flow Transport State Tracking:** Extend the current monitoring framework to maintain transport state on a per-flow basis, enabling finer-grained correlation between transport behavior and RAN dynamics.
- **QUIC Support:** Extend RATO's telemetry and control mechanisms to support QUIC, which is becoming increasingly prevalent in modern Internet applications.
- **Accurate ECN (AccECN) Integration:** Leverage AccECN to provide richer congestion feedback and establish a more direct feedback loop between transport endpoints and the RAN.
- **Adaptive ECN/L4S Marking:** Develop more sophisticated marking algorithms that account for factors such as user mobility, rapidly varying radio conditions, short-RTT flows, and flow-specific characteristics.

- **Dynamic RLC Buffer Management:** Utilize real-time telemetry to dynamically adjust RLC buffer sizes, improving the latency-throughput trade-off under varying network conditions.
- **Transport-Aware Scheduling and Resource Allocation:** Incorporate transport-level information into MAC-layer scheduling and radio resource allocation decisions to optimize metrics such as latency, throughput, and flow completion time.

References

[1] Saleh Abdullah et al. “Improving the TCP Newreno Congestion Avoidance Algorithm on 5G Networks”. In: *Journal of Communications* (2023), pp. 228–235. DOI: 10.12720/jcm.18.4.228-235.

[2] Saif E. A. Alnawayseh, Waleed T. Al-Sit, and Taher M. Ghazal. “Smart Congestion Control in 5G/6G Networks Using Hybrid Deep Learning Techniques”. In: *Complexity* (2022), p. 1781952. DOI: https://doi.org/10.1155/2022/1781952.

[3] Omar Imhemed Alramli et al. “MSS-TCP: A congestion control algorithm for boosting TCP performance in mmwave cellular networks”. In: *ICT Express* (2025), pp. 631–635. DOI: 10.1016/j.icte.2025.05.005.

[4] Omar Imhemed Alramli et al. “RTTV-TCP: Adaptive congestion control algorithm based on RTT variations for mmWave networks”. In: *Ad Hoc Netw.* (2024). DOI: 10.1016/j.adhoc.2024.103611.

[5] Davide Brunello et al. “Low Latency Low Loss Scalable Throughput in 5G Networks”. In: *VTC ’21-Spring*. IEEE, 2021, pp. 1–7. DOI: 10.1109/VTC2021-Spring51267.2021.9448764.

[6] Jacky Cao, Xiang Su, and Pan Hui. “Towards Optimising Transport Protocols on the 5G Edge for Mobile Augmented Reality”. In: *IXR ’23*. Ottawa ON, Canada: ACM, 2023, pp. 3–9. DOI: 10.1145/3607546.3616806.

[7] Xenofon Foukas. *srsRAN_Project_jbpf*. 2025. URL: https://github.com/xfoukas/srsRAN_Project_jbpf.

[8] Xenofon Foukas et al. “Taking 5G RAN Analytics and Control to a New Level”. In: *MobiCom ’23*. New York, NY, USA: ACM, 2023. DOI: 10.1145/3570361.3592493.

[9] Srihari Gopinath et al. “Improving TCP Performance via Enhanced PDCP Reordering in 5G and Beyond Networks”. In: 2024, pp. 01–06. DOI: 10.1109/WCNC57260.2024.10570592.

[10] Lingfeng Guo et al. “Stateful-BBR - An Enhanced TCP for Emerging High-Bandwidth Mobile Networks”. In: *IWQOS’21*. IEEE/ACM, 2021, pp. 1–9. DOI: 10.1109/IWQOS52092.2021.9521358.

[11] Mikel Irazabal and Navid Nikaein. “TC-RAN: A Programmable Traffic Control Service Model for 5G/6G SD-RAN”. In: *IEEE Journal on Selected Areas in Communications* (2024), pp. 406–419. DOI: 10.1109/JSAC.2023.3336162.

[12] Madhan Raj Kanagarathinam et al. “NexGen D-TCP: Next Generation Dynamic TCP Congestion Control Algorithm”. In: *IEEE Access* (2020), pp. 164482–164496. DOI: 10.1109/ACCESS.2020.3022284.

[13] Fatemeh Kavehmadavani et al. “Empowering Traffic Steering in 6G Open RAN With Deep Reinforcement Learning”. In: *IEEE Transactions on Wireless Communications* (2024), pp. 12782–12798. DOI: 10.1109/TWC.2024.3396273.

[14] Sulaiman Khan et al. “Efficient and reliable hybrid deep learning-enabled model for congestion control in 5G/6G networks”. In: *Comput. Commun.* (2022), pp. 31–40. DOI: 10.1016/j.comcom.2021.11.001.

[15] A. N. Krasilov et al. “Performance Evaluation of TCP Data Transmission in 5G mmWave Networks”. In: *Journal of Communications Technology and Electronics* (2020), pp. 735–740. DOI: 10.1134/S1064226920060194.

[16] Swaraj Kumar et al. “Cognitive RAN-Aware Transport Layer for NR-V2X Communications in 5G and Beyond Networks”. In: *VTC ’24 Fall)*. 2024, pp. 1–7. DOI: 10.1109/VTC2024-Fall63153.2024.10757789.

[17] Andrea Lacava et al. “Programmable and Customized Intelligence for Traffic Steering in 5G Networks Using Open RAN Architectures”. In: *IEEE Transactions on Mobile Computing* (2024), pp. 2882–2897. DOI: 10.1109/TMC.2023.3266642.

[18] lance. *xfr: A modern iperf3 alternative with a live TUI, multi-client server, and QUIC support. Built in Rust.* 2026. URL: https://github.com/lance0/xfr.

[19] S. Malini and P. Karthigaikumar. “A Weighted AIMD Congestion Control Algorithm for Device-to-Device Communication in 5G Network”. In: *IETE Journal of Research* (2025), pp. 131–138. DOI: 10.1080/03772063.2024.2400257.

[20] Weskley Mauricio, Francisco Hugo Costa Neto, and Maykon R. Pereira Da Silva. “Performance Analysis of Transport Protocols and RLC modes in 5G Open RAN Networks”. In: *WONS’ 25*. 2025, pp. 1–4.

[21] Microsoft. *jbpf: Userspace eBPF instrumentation and control framework for deploying control and monitoring functions in a secure manner*. 2025. URL: https://github.com/microsoft/jbpf.

[22] Microsoft. *JRT Controller: Real time controller for network functions instrumented with the jbpf framework*. 2025. URL: https://github.com/microsoft/jrt-controller.

[23] Van-Dinh Nguyen et al. “Network-Aided Intelligent Traffic Steering in 6G O-RAN: A Multi-Layer Optimization Framework”. In: *IEEE Journal on Selected Areas in Communications* (2024), pp. 389–405. DOI: 10.1109/JSAC.2023.3336183.

[24] Guangjin Pan, Shugong Xu, and Pin Jiang. “Optimizing 5G-Advanced Networks for Time-critical Applications: The Role of L4S”. 2024. DOI: 10.48550/arXiv.2407.20852.

[25] Pasha Pazhooheshy, Soheil Abbasloo, and Yashar Ganjali. “Reminis: A Simple and Efficient Congestion Control Scheme for 5G Networks and Beyond”. In: *IFIP Networking’ 25*. 2025.

[26] Ivan Petrov and Toni Janevski. “Advanced 5G-TCP: Transport protocol for 5G mobile networks”. In: *CCNC’ 2017*. Las Vegas, NV: IEEE Press, 2017, pp. 103–107. DOI: 10.1109/CCNC.2017.7983089.

[27] Lukas Prause and Mark Akselrod. *TCP ROCCET: An RTT-Oriented CUBIC Congestion Control Extension for 5G and Beyond Networks*. 2025. URL: https://arxiv.org/abs/2510.25281.

[28] IO Visor Project. *uBPF: Userspace eBPF VM*. 2025. URL: https://github.com/iovisor/ubpf.

[29] Sushila Seshasayee et al. “Scout: User Plane Telemetry for NextG Cellular Networks”. In: *HotMobile ’26*. New York, NY, USA: Association for Computing Machinery, 2026, pp. 19–24. DOI: 10.1145/3789514.3792048.

[30] Jangwoo Son et al. “L4S Congestion Control Algorithm for Interactive Low Latency Applications over 5G”. In: *ICME ’23*. IEEE, 2023, pp. 1002–1007. DOI: 10.1109/ICME55011.2023.00176.

[31] srsRAN. *srsRAN_Project*. 2025. URL: https://github.com/srsran/srsRAN_Project.

[32] Mikito Takahashi and Akihiro Nakao. "Cross-Layer Congestion Control Algorithm and Architecture for Communication Performance Optimization in Millimeter-Wave Networks". In: *IEEE Access* (2026), pp. 35823–35837. DOI: 10.1109/ACCESS.2025.3601170.

[33] Arno Troch et al. "Optimizing Radio Resource Allocation in 5G using Transport Network-Aware Intelligent RAN". In: *LATINCOM '24*. 2024, pp. 1–6. DOI: 10.1109/LATINCOM62985.2024.10770646.

[34] Srutha Keerthi Varala, Hosam El-Ocla, and Vikram Singh. "TCP DCERL+: Improving congestion control in mobile ad hoc networks". In: *Array* (2025), p. 100472. DOI: 10.1016/j.array.2025.100472.

[35] Haoran Wan and Kyle Jamieson. "L4Span: Spanning Congestion Signaling over NextG Networks for Interactive Applications". In: *CoNEXT '25*. New York, NY, USA: ACM, 2025. DOI: 10.1145/3768972.

[36] Yi Xie et al. "Yinker: A flexible BBR to achieve the high-throughput and low-latency data transmission over Wi-Fi and 5G networks". In: *Computer Networks* (2023), p. 109530. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2022.109530.

[37] Jinlin Xu et al. "A Modified TCP BBR to Enable High Fairness in High-Speed Wireless Networks". In: *Future Internet* (2024). DOI: 10.3390/fi16110392.