Dissecting the StarLink: Characterizing Queuing and Flow Dynamics in the Starlink Network

Hendrik Cech
Technical University of Munich
Germany

Nitinder Mohan
Delft University of Technology
Netherlands

Jörg Ott
Technical University of Munich
Germany

Abstract

Starlink has become the largest commercial LEO satellite network, yet little is known about its internal queue management and bandwidth allocation mechanisms. Prior measurement studies have documented performance variations but lack the granularity to explain the underlying causes. We present the first microscopic characterization of Starlink's transmission behavior, using controlled measurements from multiple terminals to capture per-packet dynamics at microsecond precision. We find that Starlink employs head-drop queuing rather than tail-drop, with capacities of approximately 1500 and 4000 packets on downlink and uplink, respectively. Bandwidth allocation is demand-driven, starting from a baseline of 100/30 Mbps on the downlink and uplink that ramps up by 3.4×/2× over 400 ms when flows sustain queue pressure. Active queue management aggressively induces packet loss to control queue occupancy, especially on the uplink. These mechanisms reset every 15 seconds during Starlink's reconfiguration cycle. We also find flow-level queuing that isolates latency between concurrent flows while coupling their loss on the downlink. These findings reveal that Starlink's queue management creates fundamentally different operating conditions than terrestrial networks.

CCS Concepts

• **Networks** → **Mobile networks**; **Transport protocols**; **Network experimentation**; **Network measurement**; **Network performance modeling**.

Keywords

Starlink, Satellite Network, LEO, Network Measurements, Network Modeling, Queuing, AQM, Congestion Control

ACM Reference Format:
Hendrik Cech, Nitinder Mohan, and Jörg Ott. 2026. Dissecting the StarLink: Characterizing Queuing and Flow Dynamics in the Starlink Network. In *ACM SIGCOMM 2026 Conference (SIGCOMM '26), August 17–21, 2026, Denver, CO, USA*. ACM, New York, NY, USA, 21 pages. https://doi.org/10.1145/3789240.3829162

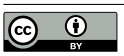
1 Introduction

Low Earth Orbit (LEO) satellite constellations represent a fundamental shift in how we deliver Internet connectivity to underserved regions. SpaceX's Starlink has emerged as the dominant commercial LEO network, currently operating approximately 9,600 satellites to serve 10.3 million subscribers across 164 countries [35, 36, 48]. With round-trip times as low as 20–40 ms, Starlink makes latency-sensitive applications viable over satellite links for the first time.

This scale and performance have attracted significant research attention. Early measurement studies established that Starlink can deliver throughput exceeding 100 Mbps while maintaining sub-50 ms latencies under favorable conditions [35]. Subsequent work revealed that these metrics vary substantially over time, with performance tied to a globally synchronized 15-second reconfiguration interval (RI) during which bandwidth and latency can shift dramatically [23, 36, 51]. More recent studies have examined Starlink's latency characteristics in detail [20] and inferred properties of its physical-layer transmission from packet timing [19]. However, these analyses operate at timescales of seconds to minutes, capturing aggregate behavior but missing the sub-second dynamics that determine how individual transport flows perform. The transport-layer behavior of Starlink remains another puzzle. Garcia et al. [21] provide the most comprehensive congestion control evaluation to date, benchmarking 14 algorithms and finding that BBR [6] consistently outperforms CUBIC [26], Linux's default congestion control algorithm. Recent concurrent work has identified drop-front buffer management as one contributing factor [22], but the interplay between queuing policies, bandwidth allocation, and congestion control on both up- and downlink remains unexplored.

Designing effective (transport and application) protocols and network management strategies for LEO require understanding the link's internal behavior, specifically its queuing policies, scheduling, and bandwidth management. Starlink's proprietary architecture, however, remains a black box, leaving protocol designers to infer mechanisms from aggregate performance alone. In this paper, we address this gap through controlled microscopic measurements from multiple Starlink dishes across Europe and the United States. We developed `NetScalpel`, a custom measurement tool that captures per-packet timestamps with microsecond precision (Figure 1), and designed targeted experiments to uncover the queuing behavior of Starlink from multiple perspectives. Our measurements reveal three transmission mechanisms of Starlink.

(1) Starlink employs head-drop queuing rather than the conventional tail-drop policy (§4.1). When the buffer fills, the packet at the front of the queue is discarded instead of rejecting new arrivals. This policy reduces average queuing delay but produces loss patterns that loss-based congestion controllers do not expect. We measure queue capacities of approximately 1500 packets on the downlink and 4000 packets on the uplink.
(2) Starlink employs demand-driven bandwidth allocation (§4.2). A baseline capacity of approximately 100 Mbps on the downlink and 30 Mbps on the uplink is available immediately. When demand exceeds this baseline, Starlink ramps up the allocation



This work is licensed under a Creative Commons Attribution 4.0 International License.
SIGCOMM '26, Denver, CO, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2467-1/26/08
https://doi.org/10.1145/3789240.3829162

over approximately 400 ms, increasing capacity by up to 3.4× on the downlink and 2× on the uplink. Conservative congestion control strategies that avoid building queues will not be able to claim their full share of capacity over Starlink.

(3) Starlink operates active queue management (AQM) that deliberately induces packet loss to control queue occupancy (§4.3). The uplink AQM is aggressive, preventing flows from filling even a quarter of the physical queue capacity. The downlink AQM is more permissive but still drops packets well before the buffer saturates. Loss-based protocols interpret these drops as congestion signals and reduce their rate, precisely when Starlink needs sustained queue pressure to maintain elevated bandwidth allocation.

These three mechanisms interact with Starlink's 15-second reconfiguration cycle (§4.4). Each reconfiguration resets the bandwidth allocation, forcing flows to re-probe for capacity even when the satellite link itself does not change. Model-based congestion controllers like BBR succeed on Starlink because they probe for bandwidth independently of loss, tolerating the AQM drops that cause CUBIC to back off (§4.6). We also find evidence of flow-level queuing that decouples one-way delay between concurrent flows, though loss rates remain coupled on the downlink due to the shared buffer.

Our work characterizes Starlink's queuing policies, bandwidth allocation mechanisms, and their interaction with transport protocols on both up- and downlink at fine-granular timescales. We hope that these insights will inform the design of protocols that can better harness Starlink's capabilities, and provide a foundation for modeling and emulating LEO satellite networks more accurately. To foster reproducibility and enable future research, we make our measurement tool `NetScalpel` [9], datasets [11], analysis code [8], and supporting software [10] publicly available.

Ethical Considerations. We measured from dedicated dishes on residential subscriptions with no cross-traffic. The residential Starlink subscription caps the resource consumption of our measurements; per-UT bandwidth allocation (§4.2) and flow isolation (§4.5) further bound the effect on neighbors.

2 Background & Related Work

Starlink subscribers connect to the network through a user terminal (UT) that communicates with Low Earth Orbit (LEO) satellites via Ku-band radio links (Figure 1). The satellites relay traffic to ground stations (GSs, also called "gateways") over Ka-band connections using multiple transceivers, which in turn connect to Starlink Points of Presence (PoPs) where the subscriber's IP address is assigned and traffic enters the public Internet. In this architecture, the uplink (UL) queue resides at the UT while the downlink (DL) queue resides on the serving satellite; both are governed by a MAC scheduler that allocates bandwidth through a grant-based mechanism [51].

Early studies established Starlink's ability to deliver broadband performance with median RTTs below 50 ms and throughput exceeding 100 Mbps [35]. Large-scale measurements using M-Lab tests and RIPE Atlas probes revealed globally synchronized 15-second reconfiguration intervals that induce periodic latency and throughput fluctuations [23, 36]. Studies operating at timescales of seconds to minutes have characterized latency dynamics and uplink delay variations [20], inferred physical-layer transmission rates from packet timing [19], and developed throughput prediction models [34]. Concurrent work by Garcia et al. [22] identified drop-front buffer management on Starlink's downlink, a finding our measurements corroborate, and reported an absence of per-flow fair queuing, which our measurements contradict. Our work further characterizes demand-driven bandwidth allocation, active queue management, and their combined impact on transport-layer performance on both up- and downlink.

Starlink Operations. SpaceX patents on Medium Access Control (MAC) scheduling [28], Radio Link Control (RLC) [24], and handover management [12] provide some insight into Starlink's design. Starlink's layer-2 comprises a MAC sub-layer and RLC sublayer, both operating on satellites. The MAC scheduler allocates transmission grants per beam in 1.33 ms frames [23, 27], adapting modulation and coding schemes (MCS) based on Signal-to-Noise Ratio (SNR) and Block Error Rate (BLER) feedback from UTs. Uplink (UL) grant allocation prioritizes UTs with low bandwidth demand while assigning unused capacity to high-demand terminals. The allocation relies on each UT reporting its buffer occupancy to the MAC scheduler, enabling demand-driven resource distribution. The RLC layer manages per-flow queues where Quality of Service (QoS) or AQM mechanisms like RED may be applied [24]. Handovers follow pre-computed schedules, with duplicate packet transmission employed to prevent loss during satellite or ground station transitions [12]. Understanding how these architectural choices affect end-to-end performance has motivated extensive measurement studies [19, 20, 23, 34–36]. Garcia et al. [21] provide the most comprehensive evaluation of TCP performance over Starlink, benchmarking 14 congestion controls and finding that BBR consistently outperforms CUBIC. Their analysis also revealed a first-comer advantage in flow fairness. To mitigate performance degradation during Starlink's reconfigurations, several proposals preemptively stall transmissions or freeze the congestion window [30, 31, 52], while others design LEO-aware CCAs [32].

Open Questions. While patents mention per-flow queues and RED, the queue management strategy, drop policy, and bandwidth allocation behavior in production remain unknown. Prior studies have characterized Starlink's latency, throughput, and handover patterns but have not systematically evaluated sub-second bandwidth dynamics, queuing policies, or their combined impact on congestion control. *Our work addresses these gaps through targeted measurements.* We reveal that bandwidth ramps up over ≈ 400 ms when demand exceeds the baseline allocation and resets at each 15-second reconfiguration. The bottleneck queues reside on the UT for UL and on the serving satellite for DL, with capacities of ≈ 4000 packets (UL) and ≈ 1500 packets (DL). Starlink uses head-drop rather than tail-drop queuing and deliberately induces losses to control queue occupancy, consistent with AQM. Finally, we show evidence of flow-level queuing that provides latency isolation and analyze how these mechanisms collectively affect transport performance.

3 Measurement Methodology

Understanding Starlink's queuing and bandwidth dynamics requires precise measurements that existing tools do not adequately support. Existing throughput measurements focus on a single task,

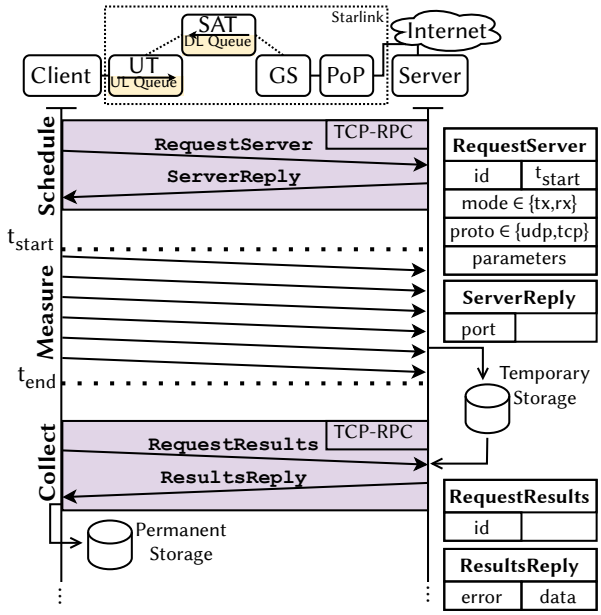


Figure 1: NetScalpel architecture. A client behind the Starlink user terminal (UT) and a cloud-hosted server coordinate UDP and TCP experiments, recording per-packet timestamps at both endpoints.

e.g., conducting throughput tests with `iperf3` and latency measurement with `iRTT`. For our purpose, an integrated tool is required to precisely schedule different types of measurements in conjunction. To address these limitations, we developed a custom measurement framework that provides fine-grained control over packet transmission timing, supports synchronized scheduling around Starlink's reconfiguration events, and captures per-packet metrics for detailed post-hoc analysis.

3.1 Measurement Infrastructure

Measurement Tool. We developed `NetScalpel`, a custom network measurement tool designed for fine-grained control over packet transmission timing and scheduling (Figure 1). The tool is written in Go and consists of a client and server component that coordinate test execution and exchange results over a TCP control channel using RPC calls. Both components record per-packet send and receive timestamps with microsecond precision, enabling post-hoc computation of one-way delay (OWD), packet loss, and receive rate. The analysis relies on NTP synchronization between endpoints.

`NetScalpel` supports three categories of UDP-based measurements, each targeting different aspects of Starlink's behavior. *Fixed-rate* measurements transmit UDP packets at a configurable send rate, duration, and packet size, allowing us to probe the link's response to sustained load and observe bandwidth adaptation over time. *Periodic probes* send individual UDP packets at regular intervals (e.g., one packet per millisecond) to measure baseline OWD on lightly loaded links without inducing queuing. *Burst mode* transmits a configurable number of packets at line rate using the Linux `sendmmsg` system call, enabling investigation of queue capacity and drop policies by overwhelming the queue instantaneously. The use of UDP for targeted measurements allows us to finely control the transmission of data and observe the network behavior on a per-packet basis. We validated the behavior of our tool by confirming the generated traffic patterns with `pcap` traces.

TCP Measurements. `NetScalpel` also orchestrates TCP measurements with configurable parameters such as duration, rate, and congestion control algorithm (CCA). The tool collects kernel-level metrics every 5 ms via the `tcp_info` socket option, including congestion window size, packets in flight, smoothed RTT, and retransmission counts. We set up a custom Linux kernel (6.13.7) that merges implementations of different CCAs to allow for direct and fair comparison[1]. For per-packet TCP analysis, we built a Rust tool that matches packets across sender and receiver captures using sequence number, acknowledgment number, flags, and timestamp. We disabled TCP Segmentation Offload (TSO) and Large Receive Offload (LRO) to ensure MTU-sized packets on both ends, and validated reliable packet matching during load phases.

Experiments. Within `NetScalpel`, we defined *experiments* that schedule the execution of a specific set of UDP- or TCP-based measurements ahead of time, typically for the next reconfiguration interval (RI). This allowed the precise execution of network tests without mixing measurement and control traffic. Specifically, we designed eight experiments, which are detailed in Table 4 in the Appendix. `MultiDurationRate` sends UDP packets at varying rates and durations to characterize bandwidth allocation (§4.2) and AQM behavior (§4.3); `Burst` sends packets at line rate to probe queue capacity and drop policy (§4.1); `Cooldown` measures how long elevated bandwidth persists after a flow stops (§4.2); `SwitchFlow` tests whether bandwidth allocation is per-UT by handing off between flows (§4.2); `Rate` runs long-duration saturating transfers spanning multiple RIs (§4.4); `OWD` probes idle-link delay across RIs (§4.4); `MouseElephant` runs two concurrent flows to investigate flow-level queuing (§4.5); and `TCPReconf` compares CCAs across RIs (§4.6).

Experimental Setup. Our primary vantage point (VP) was a controlled residential Starlink installation in Munich, Germany, served by both 42° and 53.2° Starlink constellations using a second-generation circular dish (hardware revision `rev2_proto3`). The measurement server was hosted on AWS in Frankfurt (eu-central-1), located close to our assigned Starlink PoP. Our tests revealed a low-latency path (typically 3–5 ms) to the PoP with sufficient bandwidth, minimizing the contribution of the wired Internet path to our measurements. The client machine was connected to the Starlink dish via Gigabit Ethernet, ensuring that the local access link did not introduce additional bottlenecks or queuing. We calibrated the clocks of the endpoints with high-accuracy stratum 1 NTP servers that were located within an RTT of less than 5 ms [14]. These provided sufficient accuracy for our analysis of queue dynamics on the order of milliseconds. We verified that the client's network stack did not limit our measurements by configuring appropriately sized socket buffers and disabling any local traffic shaping.

To verify that the observed behaviors are inherent to Starlink rather than artifacts of our specific location, we replicated key experiments from dishes on other continents. We leveraged the

[1] Source code taken from [15, 25, 44, 47, 49, 50].

LEOScope platform [16] for measurements from the US west coast, and obtained access to a dish on the US east coast. Both VPs were served by the same constellations as our primary VP, albeit mapped to different GSs and PoPs (Seattle and New York, respectively).

3.2 Data Collection

Starlink's network undergoes periodic *reconfigurations* at globally synchronized times: seconds 12, 27, 42, and 57 of each minute [21, 36, 51]. Our measurement campaign comprises three test categories designed to probe different aspects of Starlink's behavior within and across these intervals. For queue and bandwidth characterization, we conducted fixed-rate UDP tests with send rates ranging from 1–700 Mbps on the downlink (DL) and 1–200 Mbps on the uplink (UL), with durations up to 4000 ms. To saturate the link and probe its maximum capacity, we selected 500 Mbps on the DL and 140 Mbps on the UL for most experiments, values that consistently exceeded the observed available bandwidth. Burst tests transmitting 3,000 packets on the DL and 10,000 packets on the UL with payload sizes of 0, 700, and 1400 bytes allowed us to determine queue capacity and drop policy. Finally, 60 s tests spanning multiple RIs captured the impact of reconfigurations on established flows.

We randomized test order and scheduled them with a one-second gap to each reconfiguration point to ensure robustness and avoid measurement artifacts from residual queue state. We repeated each experiment configuration between 30 and 130 times at our primary VP (see Table 3 in the Appendix for exact counts per experiment). Throughout this paper, we report 95% confidence intervals (shown as shaded regions) and verify that our sample sizes yield statistically significant results. We consistently use **blue** for downlink (DL) and **orange** for uplink (UL) results. Our measurement campaign ran from June 2025 to February 2026; we validated our core findings by repeating experiments across VPs throughout the measurement campaign. TCP experiments were excluded as they require a custom kernel deployment at each VP. Our final dataset [11] consists of ≈ 400 GB of measurements, comprised of compressed packet captures, per-packet timestamps, and other metadata.

Satellite Identification. To correlate network behavior with satellite handovers, we identified which satellite the dish was connected to during each measurement. The Starlink dish exposes an *obstruction map*, a 123 × 123 pixel grid representing the UT's view of the sky that we fetch once per second. The difference of adjacent maps shows at which angle the connected satellite was positioned at that point in time. To identify the serving satellite, we downloaded current Two-Line Element (TLE) data from CelesTrak and computed predicted satellite trajectories from our dish location. The serving satellite is the one with the trajectory that best matches the observed motion. Our approach follows the methodology and implementation of Ahangarpour et al. [1, 2].

3.3 Queue State Estimation

Understanding Starlink's queuing dynamics requires estimating two key metrics from our per-packet measurements: *sojourn time* (time spent waiting in the queue) and *queue occupancy* (number of packets in the queue). We developed the event-driven reconstruction Algorithm 1 that models the queue's behavior by tracking when packets enter and exit the bottleneck queue based on per-packet send and receive timestamps.

Algorithm 1 Head-Drop Queue State Estimation

Require: Set of packets P. Each $p \in P$ has:
p_{tx} (send time), p_{rx} (receive time), status $p_s \in \{\text{rcvd}, \text{lost}\}$.
Ensure: Queue occupancy time series Q.

▷ *Event list, propagation delay estimate, and sojourn times of received pkts*
1: $\mathcal{E} \leftarrow$ List()
2: $d_{\min} \leftarrow \min((p_{\text{rx}} - p_{\text{tx}}) \mid p \in P, p_s = \text{rcvd}\})$
3: $D_{\text{sojourn}} \leftarrow \{(p_{\text{rx}} - p_{\text{tx}}) - d_{\min} \mid p \in P, p_s = \text{rcvd}\}$

4: **for all** $p \in P$ **do**
5: Add $(p_{\text{tx}}, +1)$ to $\mathcal{E}$ ▷ *Enqueue on send*
6: **if** $p_s = \text{rcvd}$ **then**
7: $t_{\text{exit}} \leftarrow p_{\text{rx}} - d_{\min}$
8: Add $(t_{\text{exit}}, -1)$ to $\mathcal{E}$ ▷ *Packet dequeued*
9: **else** ▷ *Lost: interpolate sojourn time from neighbors*
10: $d_{\text{sojourn}} \leftarrow \textsc{InterpolateSojourn}(D_{\text{sojourn}}, p)$
11: $t_{\text{drop}} \leftarrow p_{\text{tx}} + d_{\text{sojourn}}$
12: Add $(t_{\text{drop}}, -1)$ to $\mathcal{E}$ ▷ *Packet dropped from head*
13: **end if**
14: **end for**

15: Sort $\mathcal{E}$ by timestamp t
16: $q \leftarrow 0$; $Q \leftarrow$ List() ▷ *Running queue occupancy*

17: **for all** $(t, \Delta q) \in \mathcal{E}$ **do**
18: $q \leftarrow q + \Delta q$
19: Append (t, q) to Q
20: **end for**
21: **return** Q

The minimum observed OWD is an estimate of the propagation delay (line 2). We calculate sojourn time by subtracting the estimated propagation delay from each packet's measured OWD (line 3). For queue occupancy, we reconstruct the queue state by simulating three types of events that change the number of packets in the queue. Each transmitted packet generates an *enqueue event* at its send timestamp, incrementing the queue size by one (line 5). Successfully received packets generate *dequeue events* at their estimated queue exit time (line 8). We compute this exit time by subtracting the propagation delay from the packet's reception timestamp, which gives us the approximate moment the packet left the queue (line 7). For lost packets, we leverage our finding that Starlink employs head-drop queuing (§4.1), meaning dropped packets wait in the queue before being discarded rather than being rejected upon arrival. Since we cannot directly measure the OWD of a lost packet, we estimate its sojourn time by interpolating between the sojourn times of neighboring received packets (line 10). This interpolated sojourn time allows us to estimate when the packet was dropped from the queue head, creating a *drop event* that decrements the queue size (line 12). By sorting all enqueue, dequeue, and drop events chronologically and computing the running sum (adding one for each enqueue, subtracting one for each dequeue or drop), we reconstruct the queue occupancy over time.

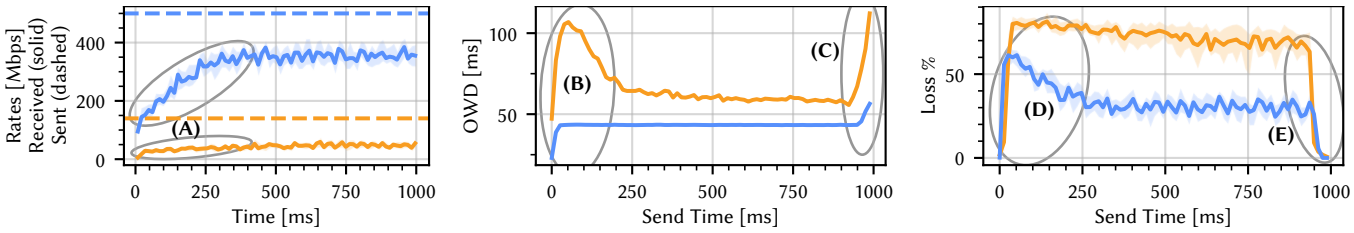


Figure 2: We saturate DL (blue) and UL (orange) with 500/140 Mbps for 1 s and plot median bandwidth, OWD, and loss across 100 measurements. These show bandwidth ramp-up (A), decreasing UL OWD (B) and loss (D), and non-monotonic patterns at transfer end (C, E). We attribute these behaviors to demand-driven bandwidth allocation [(A), (B), (D), §4.2], deliberate packet drops [(B), §4.3], and head-drop queue management [(C), (E), §4.1].

This approach has five limitations. First, it relies on sufficiently synchronized clocks (we use NTP). Second, we assume packets are immediately enqueued after sending and that propagation delay occurs entirely between queue and receiver. On the UL, this holds since the client connects directly to the dish via Ethernet. On the DL, queue arrival is consistently estimated 3–5 ms too early due to the (invisible) Starlink's ground station connection, though this constant offset does not affect our analysis. Third, a drifting propagation delay baseline inflates the estimated queue size. This drift stems from mobility within the Starlink network and primarily occurs at RIs. We therefore apply queue estimation only to short measurements (up to 4 s) within a single RI, except for the TCP measurements in §4.6. Fourth, the minimum OWD is only an upper bound of the propagation delay due to potential cross-traffic. Starlink's UT isolation, flow queuing (§4.5), and our exclusive dish access however tighten this bound. Fifth, we attribute all losses to head-drop actions, though some may stem from transmission errors which would inflate queue size estimates. An upper bound on Starlink's RF-only loss is the idle-link loss rate (Figure 18) of 0.03/0.02% on the DL/UL, respectively. A misattributed loss inflates the queue estimate only during the packet's sojourn time.

4 Starlink Characterization

We now turn to characterizing Starlink's bandwidth, latency, and loss dynamics at sub-second timescales. Our measurements reveal several behaviors that are not visible in prior studies focused on longer timescales. Figure 2 shows the median of 80 one-second saturating UDP transfers on both DL (blue, 500 Mbps send rate) and UL (orange, 140 Mbps send rate). Five phenomena consistently appear: The receive rate gradually increases over the first ≈ 400 ms despite a constant send rate (see (A) in the figure). During the first ≈ 100 ms, both OWD and loss rate rise steeply before declining to a lower, stable level (see (B) and (D)). Towards the end of each transfer, the OWD increases roughly linearly while the loss rate drops sharply (see (C) and (E)). In the following sections, we attribute these behaviors to three mechanisms that we uncover: head-drop queue management (see §4.1), demand-driven bandwidth allocation (see §4.2), and deliberate packet drops consistent with AQM (see §4.3).

4.1 Queue Investigation

The latency spike (C) and loss rate drop (E) in Figure 2 suggest that Starlink employs a *head-drop* queuing strategy: when a queue reaches its capacity, the packet at the *front* is discarded to make room for the newly arriving packet, rather than rejecting the arrival as in the more common tail-drop policy. By always removing the oldest packet first, head-drop reduces the sojourn time of all queued packets. We support this hypothesis with two independent lines of evidence: burst measurements that reveal queue capacity and drop order, and controlled tests that reproduce Starlink's characteristic OWD and loss patterns.

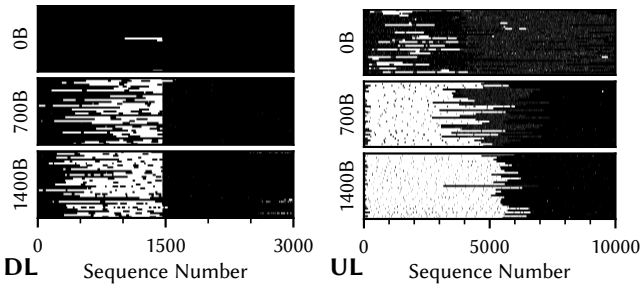


Figure 3: Receive patterns of UDP burst tests with 0, 700, and 1400 B payloads. Rows are tests; black cells mark received packets. The last-sent packets are consistently received while earlier packets are lost, indicating head-drop queuing with capacities of ≈ 1500 (DL, left) and ≈ 4000 packets (UL, right).

We use the `Burst` experiment to repeatedly send 3K/10K UDP packets on the DL/UL, respectively, with payload sizes of 0, 700, or 1400 bytes (plus an 8 byte sequence number), and record which packets arrive. The packets are *bursted*, i.e., sent at the transmission speed of the physical Gigabit Ethernet network interface using the Linux `sendmmsg` system call; the largest burst takes 36/120 ms on DL/UL. We ensure sufficiently sized buffers at the sender and verify via queue and NIC statistics that all loss occurs in the network.

Figure 3 shows the results. In both directions, the first few hundred packets are delivered before the queue fills. For the 700 and 1400 B payloads, a sharp boundary then appears: on the DL, earlier packets (below sequence number ≈ 1500) are lost while all later packets are received; on the UL, this boundary falls around sequence number 6000. This pattern is the signature of head-drop: once the queue reached capacity, each newly arriving packet displaced the

packet at the front, so the *last*-sent packets survived. From the position of this boundary, we infer a queue capacity of ≈ 1500 packets on the DL and ≈ 4000 packets on the UL.

Notably, almost no packets are dropped when the payload is 0 B. At first glance, this suggests byte-sized queues. However, the received tail blocks of 700 B and 1400 B packets have similar sizes; a byte-sized queue that fits ≈ 1500 packets of 1400 B should hold roughly twice as many 700 B packets, which we do not observe. We conclude that the queues enforce a *packet* limit. Each 0 B packet is only 36 B on the wire, so the full DL burst completes in under 1 ms at 100 Mbps and the UL burst in 9.6 ms at 30 Mbps[2]. At these rates, the queue drains faster than it fills, never reaching capacity.

To verify our hypothesis, we test both policies in a network of serially-connected Linux namespaces resembling the Starlink path (client, UT, satellite, server). The DL/UL rates are limited to 300/70 Mbps with queue sizes of 1500/4000 packets, respectively, using the HTB queuing discipline. Figure 4 compares a saturating UDP transfer (700/200 Mbps on DL/UL) under head-drop (solid black) and tail-drop (dashed black) with Starlink's measured behavior (colored). The queue size is estimated with our queue state estimation algorithm (see §3.3) and validated against ground truth captured during the test. Under tail-drop, both OWD and loss rate stabilize once the queue fills to capacity. Under head-drop, a distinctive end-of-transfer pattern emerges: once sending stops, no new packets displace queued ones, so the remaining packets drain in order without further loss and with progressively longer sojourn times, since each successive packet has waited longer in the queue. This produces the linear OWD rise and sharp loss rate drop visible in the final tens of milliseconds (α and β in Figure 4) of each transfer, e.g., $\beta = 222$ ms on the emulated head-drop UL. The OWD when sending stops approximates the queue drain duration, i.e., α ms in the Starlink measurements and β ms in the emulation. The head-drop test closely reproduces Starlink's measured end-of-transfer behavior on both UL and DL, confirming that head-drop is the underlying mechanism. In contrast, tail-drop exhibits an almost 3-fold higher sojourn time in our controlled setup.

Starlink's behavior does differ from the pure head-drop procedure in one key respect: on the UL, the OWD and loss rate peak after ≈ 100 ms and then decrease, even though the send rate continues to exceed the bandwidth. The estimated queue size confirms that this decline correlates with a decreasing number of queued packets—well before the maximum capacity of 4000 packets is reached. A similar initial increase in loss rate is visible on the DL. We attribute these deviations to two additional mechanisms: demand-driven bandwidth allocation (§4.2) and active queue management (§4.3).

> *Takeaway* — Starlink employs a head-drop strategy on both DL and UL with a queue size of 1500 and 4000 packets, respectively. This policy reduces sojourn time (and therefore one-way delay) when the queue operates at its capacity.

[2]The small UDP burst packets with a 0 B payload have a size of 36 B (IP + UDP + sequence number). Transmission time in ms: (36 B · N packets) ÷ R Bps · 100.

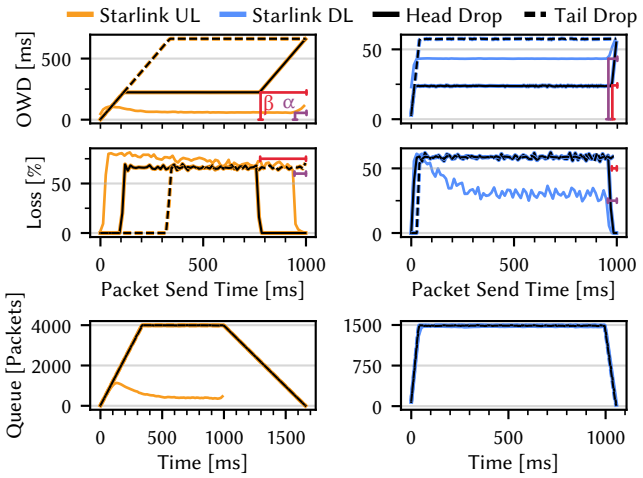


Figure 4: Saturating UDP transfers under head-drop (solid black) and tail-drop (dashed black) in a controlled environment, compared to Starlink (colored, as in Figure 2). The left shows UL and the right the DL. The head-drop test reproduces Starlink's end-of-transfer OWD rise and loss rate drop, while tail-drop yields stable OWDs and losses throughout.

4.2 Bandwidth Allocation

Even though the send rate of the tests in Figure 2 stays constant, the receive rate increases noticeably over the first ≈ 400 ms (behavior A). This implies that Starlink does not provide its full bandwidth immediately but allocates it dynamically in response to demand. The rising bandwidth also explains the concurrent OWD (behavior B) and loss rate (behavior D) drops. As the queue drains faster, sojourn time decreases and fewer packets must be dropped.

Demand-Driven Allocation. By measuring different constant send rates, the `MultiDurationRate` experiment confirms that the bandwidth allocation is demand-driven (Figure 5). A baseline bandwidth is available from the start of each transfer (≈ 100 Mbps on DL, ≈ 30 Mbps on UL). Send rates below these baselines produce a flat receive rate throughout. Higher send rates trigger a ramp-up over ≈ 400 ms that multiplies the available bandwidth by up to 3.4× on the DL (≈ 340 Mbps) and 2× on the UL before both directions settle on a plateau. This is consistent with the MAC scheduler on the satellites adapting each UT's allocation to its current demand (see §2). Synchronized micro-spikes in the receive rate across send rates further support this interpretation and hint at the scheduling interval at which the scheduler re-evaluates allocations and modulation and coding schemes (MCS).

Allocation Decay. The `Cooldown` experiment measures how long the additional bandwidth persists after a flow stops sending (Figure 6). A UDP flow is active for 800 ms (*phase 1*), pauses for a variable duration (0–600 ms in 25 ms steps), then resumes for another 800 ms (*phase 2*) at a constant send rate of 500/140 Mbps on DL/UL. We quantify the remaining allocation as the ratio of the initial receive rate in phase 2 (yellow region B) to the pre-ramp-up rate in phase 1 (blue region A). Ratio $B \div A > 1$ means the allocation persisted through the pause.

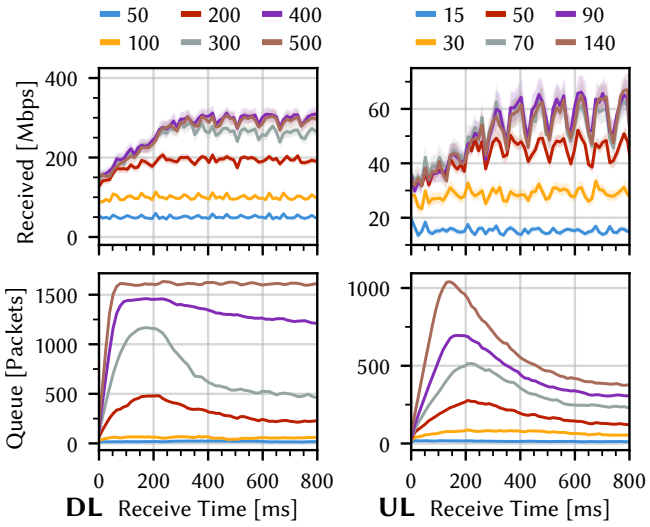


Figure 5: UDP measurements at different rates (Mbps) on the DL (left) and UL (right). A certain bandwidth is available instantaneously but higher send rates trigger a bandwidth ramp-up that takes up to 400 ms during which packets accumulate temporarily in the queue.

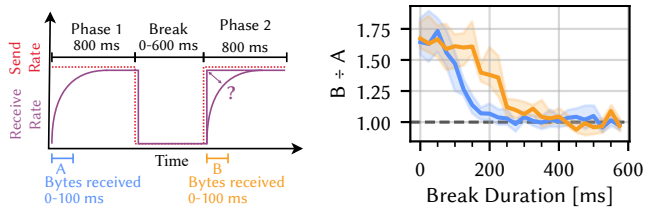


Figure 6: We examine the decay rate of bandwidth allocation with test procedure `Cooldown`. If sending stops, the allocation quickly decays.

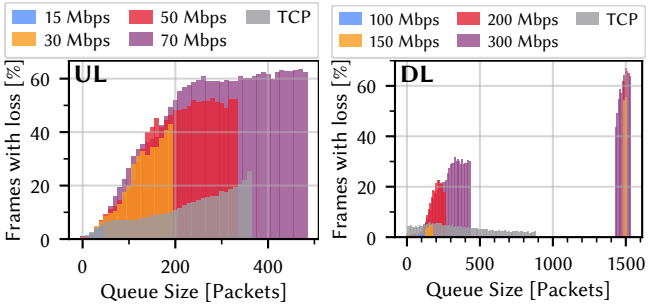


Figure 7: Fraction of 1.33 ms frames containing at least one packet loss as a function of queue size, for UDP (top) and TCP (bottom) flows at various send rates. Loss frequency rises with queue occupancy on both UL and DL, indicating deliberate, queue-aware drops.

The results reveal an asymmetric decay. The DL allocation drops roughly linearly to the baseline within ≈ 200 ms. On the UL, however, the allocation first persists for ≈ 175 ms before decaying over another ≈ 200 ms. This plateau disappears when the experiment is repeated at a lower send rate of 70 Mbps (Figure 16). *The lower rate fills the queue less, suggesting that the scheduler also considers queue occupancy when deciding how long to maintain an allocation.*

We further investigate whether the allocation is tied to individual flows or to the UT as a whole, as Starlink's patents claim per-UT allocation [28]. We design the `SwitchFlow` experiment where flow A is active for 800 ms at a rate sufficient to trigger the ramp-up before flow B takes over immediately. If the allocation were per-flow, we would expect flow B to start at the low instantaneous rate and undergo its own ramp-up, causing a visible throughput drop at the switch point. Instead, Figure 13 shows that the combined receive rate stays constant at the switch point, confirming that the bandwidth allocation is tied to the UT rather than individual flows.

> *Takeaway* — Starlink's MAC scheduler dynamically allocates bandwidth per-UT based on demand. Only a fraction of the maximum bandwidth is available immediately. Higher send rates trigger a ramp-up over ≈ 400 ms that increases the allocation by up to 3.4× (DL) and 2× (UL). The allocation decays within hundreds of milliseconds when bandwidth demand ceases.

4.3 Queue Management

The queue size traces in Figure 5 reveal a behavior that bandwidth ramp-up alone cannot explain. On the UL, the queue initially grows to ≈ 1000 packets under a saturating 140 Mbps flow. It then decreases to fewer than 500 packets, even though the send rate continuously exceeds the link's maximum average bandwidth of ≈ 60 Mbps. Similarly, on the DL, the queue starts declining under a 400 Mbps send rate that still exceeds the average maximum bandwidth by ≈ 100 Mbps. The observed decline indicates that packets are deliberately dropped to reduce occupancy; a capacity-bounded queue without deliberate drops would remain full under the observed send > receive rates. This *Active Queue Management (AQM)* mechanism explains Figure 2 (B) where the UL OWD decreases due to a shrinking queue. The AQM on the DL is less aggressive and does not cause the OWD of the high 500 Mbps send rate shown in Figure 2 to decrease. The existence of AQM is consistent with Starlink patents that mention RED as part of the RLC sub-layer (see §2). Our analysis reveals two distinct strategies. The UL AQM limits queue growth to well below the physical capacity of ≈ 4000 packets. The DL AQM is more permissive and allows the queue to approach its limit before intensifying drops.

To quantify the relationship between queue occupancy and loss, we analyze the `MultiDurationRate` flows from Figure 5. We group packets into 1.33 ms bins, matching the Starlink radio frame duration (§2), and compute the fraction of bins at each queue size that contain at least one packet loss. We exclude queue sizes observed less than 0.5% of the time to reduce noise. The UL histogram (Figure 7) shows that frame loss frequency increases roughly linearly with queue size from 0–250 packets, independent of the UDP send rate, until more than 60% of frames contain at least one loss. TCP

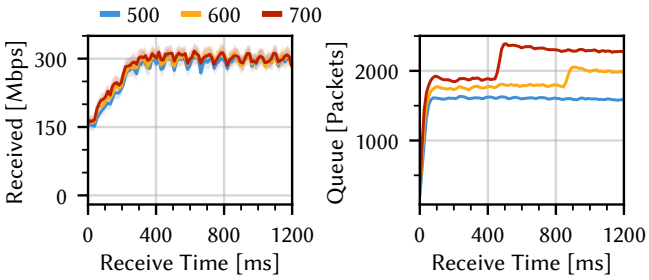


Figure 8: At DL send rates above 500 Mbps, the estimated queue size exceeds the satellite buffer capacity, suggesting additional buffering at the PoP.

flows (BBRv1, discussed in §4.6) exhibit a slower rise but still reach a frame loss rate exceeding 20% at a queue size of 300 packets.

The DL histogram (Figure 7) shows a different profile. The queue regularly reaches its capacity of ≈ 1500 packets, where the loss frequency spikes sharply. However, significant losses already occur well before saturation. The shown flows experience at least one packet loss in up to 25% of frames at low queue sizes of less than 500 packets. TCP flows exhibit a similar trend, with losses rising at queue sizes well below the physical limit, albeit less extreme than the UDP flows. The gap between ≈ 500 and 1500 packets in the DL histogram arises because intermediate queue sizes are rarely observed and fall below our 0.5% filter threshold; data points beyond 1500 are artifacts of OWD-induced overestimation.

The loss patterns in Figure 7 cannot be explained by random loss from signal interference or hardware errors. On the one hand, the observed loss frequencies on the loaded link surpass the loss on a mostly-idle link that we measure with 1 ms probes at ≈ 0.03% (Figure 18). On the other hand, random loss would produce a flat rate independent of queue size, rising sharply to 100% only at the capacity limit. Instead, loss frequency grows steadily with queue occupancy across all send rates on both UL and DL. This queue-dependent drop profile is characteristic of AQM and confirms that Starlink actively manages its queues.

Comparison with Known AQM Algorithms. Classic RED drops packets with a probability growing linearly as average queue size moves between a minimum and maximum threshold [18]. However, the loss curves shift significantly with offered load (especially between UDP and TCP), suggesting an adaptive mechanism rather than a fixed profile. A-RED extends RED by adjusting maximum drop probability based on traffic load [17], becoming more aggressive when average queue size exceeds a target. Among sojourn-time-based AQMs, CoDel drops packets on dequeue when minimum sojourn time within a window exceeds a target [38, 39]. PIE steers delay toward a setpoint [41, 42]; its adaptive aggressiveness could explain the load-dependent behavior, but PIE drops packets on enqueue, conflicting with Starlink's head-drop policy. CoDel drops on dequeue, consistent with our findings. Since queue size and sojourn time are directly related, determining which metric Starlink monitors is difficult. Plotting loss probability against sojourn time yields a similar pattern (Figure 14). While loss probability correlates with both metrics, the workload-dependent evolution shows the AQM does not simply react to absolute queue size or sojourn time. Rather, *it likely monitors their rate of change.*

DL Rate Limitation at the PoP. At DL send rates exceeding 500 Mbps, the estimated queue size grows beyond 2000 packets, far exceeding the satellite queue capacity of ≈ 1500 packets (Figure 8). We attribute this to a rate-limiting or protection mechanism at Starlink's PoP introducing additional buffering before traffic reaches the satellite link. At moderate rates, the PoP forwards traffic transparently with all queuing at the satellite. At very high rates, the PoP throttles the flow, causing packets to accumulate in a secondary upstream buffer. This aligns with DDoS protection or traffic policing, becoming manifest only at send rates well above maximum link capacity. We notified Starlink of our findings and they confirmed the use of a custom AQM and PoP-level rate limiting.

Queue Location. We attribute the observed AQM and head-drop effects to per-satellite queues for several reasons. First, our primary VP's Starlink PoP operates a 600 G peering interface [43] and GS-satellite links support 20 Gbps. These capacities are orders of magnitude larger than our peak traffic and the per-terminal Ku-band downlink capacity, making discernible queuing at the PoP or GS unlikely. Second, we observe the same behavior regardless of the time of day, making the PoP's traffic-load variations an unlikely explanation; PoP congestion would be expected to vary between peak and off-peak hours. Third, consistent head-drop and AQM-induced losses across all three VPs—traversing distinct PoPs, GSs, and user populations—suggest a mechanism common to the satellite link rather than ground infrastructure. Fourth, head-drop and AQM losses become manifest at DL rates well below 500 Mbps, clearly below the PoP rate-limiter's activation threshold. Finally, Starlink patents explicitly locate AQM within the RLC sub-layer of the radio link at the satellite (see §2).

> *Takeaway* — Starlink manages its bottleneck queues using distinct AQM strategies. The UL strategy aggressively induces losses, preventing flows from reaching even a quarter of the physical capacity. Conversely, the DL strategy is more permissive, allowing the queue to approach its maximum of ≈ 1500 packets. At very high DL rates (>500 Mbps), a PoP rate-limiter introduces additional upstream buffering.

4.4 Reconfigurations

The periodic reconfigurations of Starlink happen on a fixed 15-second "reconfiguration interval" (RI) at seconds 12, 27, 42, and 57 of each minute and induce significant bandwidth and latency fluctuations. So far, we have analyzed measurements that were conducted in-between reconfigurations to avoid interference. Our reconfiguration analysis relies on the `Rate` experiment that saturates the DL/UL with 500/200 Mbps UDP flows, respectively. Additionally, we analyze `OWD` experiment runs to observe the idle link behavior. We record the connection pattern of the Starlink dish during the experiments to infer the connected satellite at any point in time as described in §3.2. In Figure 9, we split the data by whether the reconfiguration entailed a satellite handover (solid lines) or not (dashed lines). Note that we deliberately saturate the link to probe the bandwidth limits; the reconfiguration effects are less severe at usual send rates within Starlink's capacity.

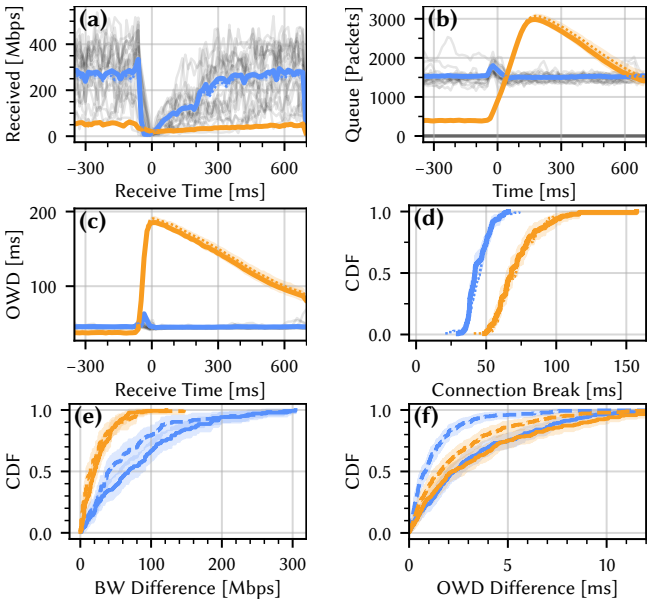


Figure 9: Reconfigurations reset the bandwidth allocation and shift the stable performance between consecutive intervals on DL (blue) and UL (orange), regardless of whether the connected satellite changed (solid) or not (dashed). (a)–(c) show receive rate, queue size, and OWD around the reconfiguration at $t = 0$; (d) the connection break duration; (e)–(f) the bandwidth and OWD shift between consecutive RIs.

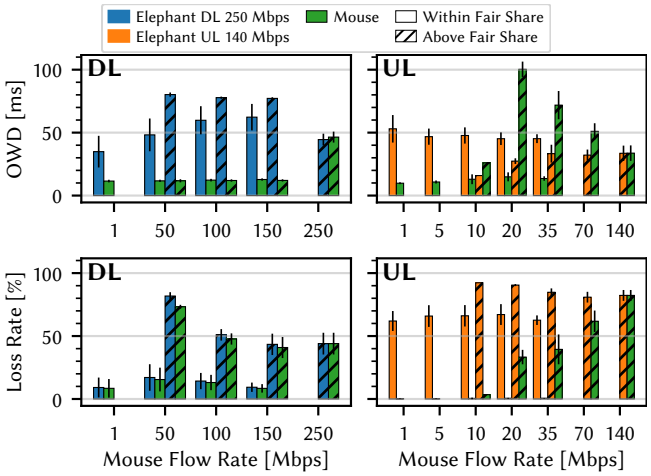


Figure 10: Two flows running concurrently over Starlink to investigate fairness: an *elephant* flow with a fixed 250 Mbps (DL) / 140 Mbps (UL) rate, and a *mouse* flow with varying rate.

Figure 9a shows the mean receive rate of the saturating UDP transfers in relation to the reconfiguration at time 0. We observe a clear drop about 100 ms before the reconfiguration point from where it recovers to the previous bandwidth over the course of ≈ 600 ms on both DL and UL. This result shows that the bandwidth allocation based on demand, as analyzed in §4.2, is redone on reconfigurations, affecting even established flows that have already been granted their share of the bandwidth. On the DL, where the queue capacity is already exhausted during normal operation, the connection break causes a temporary queue size increase of about 300 packets as well as packet loss (Figure 9b). Our algorithm estimates the total number of packets queued in the network; we argue that these packets are held at the ground station during the handover, after which they are forwarded to the satellite. On the UL, we observe a queue spike (Figure 9b) and the resulting OWD increase (Figure 9c). The deep UL queue (§4.1) is however better able to absorb the connection break than the DL queue (§A.5) since the aggressive AQM scheme keeps the queue utilization low during steady-state operation. Figure 9d shows that the schedule-driven reconfiguration results in a connection break of 45 ms on the DL and 68 ms on the UL. This break and the bandwidth allocation procedure are likely the main causes of the temporary disruption visible in Figure 9a–c.

Reconfigurations have not only temporary effects but also frequently shift the stable bandwidth and OWD between consecutive RIs. Figure 9e reveals that the median bandwidth shifts by 70 Mbps on the DL and 26 Mbps on the UL, which are ≈ 25% of the maximum measured bandwidth. We observe a similar fraction of OWD change between RIs (Figure 9f), where the OWD is measured on the idle link using the `OWD` experiment. In short, performance does not just recover but shifts to a new level after each reconfiguration.

Notably, these phenomena—the bandwidth allocation reset, the connection break duration, and the inter-RI performance shift—occur regardless of whether the connected satellite changed (dashed lines in Figure 9). This confirms the community's suspicion that the reconfiguration effects do not stem from satellite handovers but from the reconfiguration of satellite-local state.

> *Takeaway* — Reconfigurations reset the bandwidth allocation, causing a ≈ 600 ms recovery, and induce a connection break of ≈ 50 ms. Next to these temporary effects, the stable bandwidth and OWD shift between consecutive reconfiguration intervals. All of these effects occur independently of satellite handovers.

4.5 Flow Queuing

Our investigation provides evidence that Starlink employs a flow-aware queuing mechanism that provides a degree of OWD and loss decoupling between flows. Our analysis relies on the `MouseElephant` experiment with two parallel fixed-rate UDP flows on Starlink. The first *elephant* flow sends at a fixed rate of 250/140 Mbps on DL/UL, respectively, while the send rate of the parallel *mouse* flow is varied across runs. For our analysis, we classify each experimental run post-hoc using the concept of *Max-Min Fairness* [29]: We consider the mouse flow to be operating within its *fair share* if its resulting receive rate is less than half of the total aggregate (mouse + elephant) receive rate. Depending on the bandwidth available during an experiment, the same mouse send rate can be within or above its fair share. We confirm key claims of our analysis with significance tests that we detail in §B.1.

On the DL (Figure 10), we observe that the OWD of the mouse flow stays low even if the flow exceeds its fair share. This suggests a mechanism like Fair Queuing that prioritizes dequeuing packets from smaller flows. Yet, both flows experience nearly identical loss

rates, which increase as the aggregate traffic exceeds link capacity (TOST equivalence test, see §B.1). The elephant's loss rate is systematically up to 1.9 pp higher than the mouse loss rates, likely caused by the elephant flow contributing proportionally more packets to the shared queue. However, the difference remains within a ±2 pp equivalence margin across all tested rates (§B.1). A shared FIFO queue would cause the observed loss rate behavior but would also induce the same sojourn time on the two flows: Since a FIFO queue dequeues packets in order, mouse packets need to traverse the whole queue that is filled by the elephant flow, leading to the same OWDs. We believe that a form of fair queuing is realized on the DL (round-robin or similar), which leads to the observed low mouse OWDs. The coupled loss rates can be explained with the use of a single shared queue for all flows: When this buffer is full, the scheduler drops the packet at the head of the queue, regardless of which flow it belongs to. The queue size estimation confirms a shared queue limit of approximately 1500 packets (Figure 15).

Conversely, on the UL, the mouse OWD remains low as long as its traffic stays within its fair share of the link capacity. In contrast to the DL, losses are *not* coupled on the UL: the elephant suffers significantly higher loss than the mouse at every sub-fair-share rate (paired Wilcoxon signed-rank, see §B.1). The UL AQM appears to manage queue buildup and to drop packets on a per-flow basis, preventing the shared queue limit from being reached and thus avoiding the coupled loss seen on the DL.

The decoupling of OWD is advantageous for latency-sensitive applications, such as online gaming and VoIP, when sharing the link with queue-building flows. Our experiments demonstrate that the shared DL queue leads to coupled loss rates between concurrent flows when the link is saturated. An aggressive, queue-building flow can induce packet loss in a flow that operates within its fair share. Even though our tests represent a worst-case scenario due to the constant queue saturation, even irregular coupled losses can cause transport issues, e.g., due to the synchronization of TCP flows with loss-based congestion controllers. In contrast, the UL provides better flow isolation, enabled by the aggressive per-flow AQM: By managing queue buildup for each flow independently, it prevents a single aggressive flow from causing loss in others. The DL implementation of the mechanism demonstrates a strong bias towards smaller flows: For instance, a 150 Mbps mouse flow (sending at 60% of the elephant's rate) experiences a mean OWD of 12 ms (only 15% of the elephant flow's OWD of 77 ms).

> *Takeaway* — Starlink implements a flow-queuing system to decouple the OWD of competing flows, benefiting latency-sensitive applications. However, the DL's use of a shared head-drop queue couples loss between flows, failing to isolate aggressive, queue-building flows. The proactive AQM on the UL prevents that issue.

4.6 TCP Performance

The bandwidth ramp-up (§4.2), AQM behavior (§4.3), and periodic reconfigurations (§4.4), create a challenging environment for TCP Congestion Control Algorithms (CCAs). We examine how different CCAs and slow start mechanisms perform under these conditions.

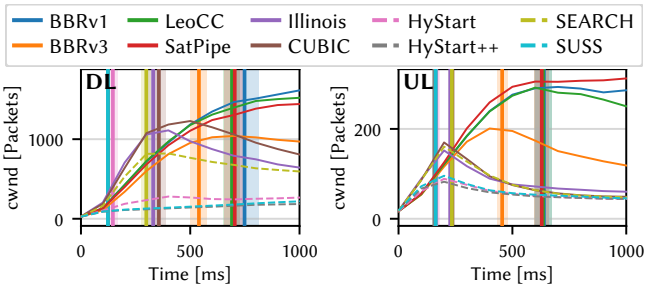


Figure 11: The congestion window during slow start on DL (left) and UL (right). The slow start exit point is marked by vertical lines. The complete runs are shown in Figure 12.

We compare six CCAs shown to work well on Starlink [21]. Of these, CUBIC [26] is loss-based, while Illinois [33] uses delay as a secondary signal. The other four are model-based algorithms: BBRv1 [6] and BBRv3 [7] build explicit network models to determine sending rates. Building directly upon BBRv1, LeoCC [32] and SatPipe [52] tailor the BBR architecture specifically for the highly dynamic conditions of Starlink. SatPipe adds an explicit queue maintenance state to BBR that preemptively drains in-flight data during deterministic handover events, leaving BBRv1's core logic otherwise unchanged. LeoCC modifies BBRv1 more deeply: it replaces BBRv1's bandwidth and RTT estimators with adaptive variants designed for Starlink's rapid capacity fluctuations, and at reconfigurations discards stale samples and limits inflight packets. We also test four slow start algorithms. Traditional exponential growth serves as our baseline. HyStart++ [4] adds a Conservative Slow Start phase, SEARCH [13] exits when delivered bytes fall below expected, and SUSS [3] accelerates window growth when predicting continued exponential growth. We conduct 12-second `TCPReconf` measurements timed to capture behavior before, during, and after reconfigurations, using saturating UDP flows as a reference of capacity. Figures 11 and 12 present the results analyzed as per the four distinct operation phases, which we discuss in turn. To rigorously confirm the performance differences discussed below, we perform pairwise significance and effect size tests between all CCAs, with methodology and results detailed in §B.2.

Phase 1: TCP Slow Start. The initial slow start phase is crucial for a CCA to quickly utilize available bandwidth. HyStart, HyStart++, and SUSS exit slow start very early as shown in Figure 11. While this prevents packet loss, it fails to capitalize on Starlink's capacity. The RTT increase from queuing normally signals congestion, but on Starlink this pressure is necessary to trigger demand-driven bandwidth allocation. In contrast, CUBIC and Illinois use classic exponential slow start and push for much larger congestion windows. CUBIC's slow start is often ended by AQM-induced loss rather than reaching queue capacity limits. The model-based CCAs (BBRv1, BBRv3, LeoCC, SatPipe) demonstrate a more balanced approach that is more cautious than CUBIC but more aggressive than HyStart. This strategy allows them to probe for capacity effectively without inducing significant loss. SEARCH takes a different approach by monitoring delivery rate similar to BBR's STARTUP phase, but we observe that it grows its window too aggressively early and lifts

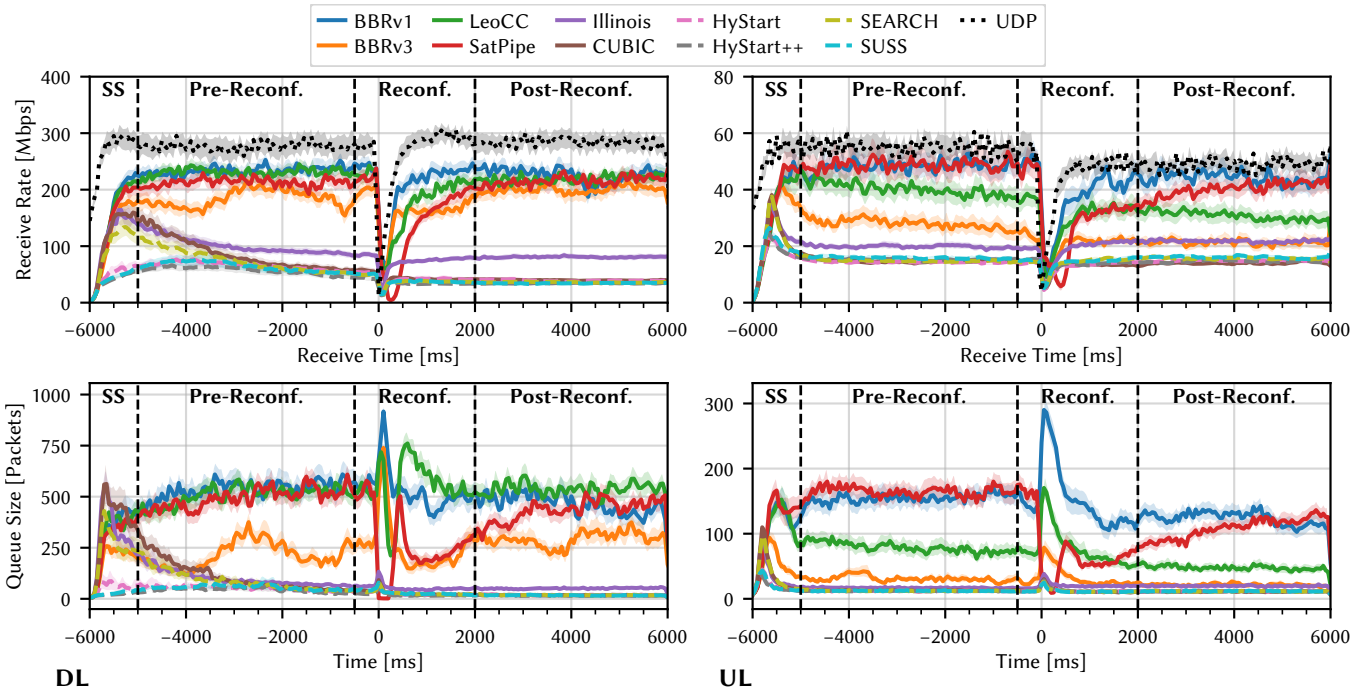


Figure 12: The performance of different TCP flavors around Starlink's reconfiguration on the DL (left) and UL (right). UDP measures the bandwidth capability of the link.

pressure prematurely. Slow start on Starlink thus requires careful balance between applying enough pressure to trigger bandwidth allocation and avoiding buffer overflow that causes AQM loss.

Phase 2: Pre-Reconfiguration. After the initial slow start, our pairwise analysis confirms a strict statistical separation: BBRv1, LeoCC, and SatPipe form a high-performance cluster that significantly outperforms the loss-based algorithms, with BBRv3 and Illinois falling squarely in between (§B.2). The loss-based CCAs see their sending rates decay due to frequent packet losses deliberately induced by Starlink's AQM. Our CUBIC tests experience a median of six packet losses in this phase, causing a progressive ramp down of the congestion window. Illinois performs slightly better than CUBIC because it uses the low OWD inflation to moderate its reaction to packet loss. Conversely, BBRv1, LeoCC, and SatPipe largely ignore packet loss and scale their sending rate based only on bandwidth estimates, allowing them to maintain rates close to the UDP baseline. BBRv3 is designed for better fairness with loss-based CCAs and refrains from probing for more bandwidth if the loss rate increases to 2%. This results in lower throughput but also maintains a lower queue size and RTT.

Phase 3: Reconfiguration. The reconfiguration (around t=0 in Figure 12) causes a temporary communication break and a reset of the bandwidth allocation, leading to a drop in TCP's delivery rate. The break itself is short (≈ 45 ms on the DL and 68 ms on the UL, see Figure 9d), but the subsequent bandwidth recovery takes much longer (at least ≈ 400 ms as shown by the saturating UDP transfer). The key issue for transport protocol performance is the reset of the bandwidth allocation. The classic loss-based CCAs are slow to reclaim bandwidth and essentially require repeating the TCP slow start phase. LeoCC's strategy of limiting in-flight packets to drain the queue after a reconfiguration is visible in the queue size plots. While it successfully decreases RTT faster than BBRv1, it also delays the ramp-up of its sending rate. We also observe frequent burst losses at the reconfiguration point on the DL but not on the UL. While the queue size grows proportionally to the sending rate during the communication break, it typically does not reach the DL queue size limit of ≈ 1500 packets. We attribute this to packet losses within the Starlink network, likely due to the AQM. SatPipe's queue maintenance state stops transmission during the handover and subsequently triggers ProbeRTT (4 MSS for 200 ms). In this way, SatPipe achieves its goal of avoiding queue build-up during the reconfiguration. The results, however, clearly show that this delays the recovery of the bandwidth allocation by several hundred milliseconds compared to the other BBR variants. The statistical tests confirm that this delayed recovery causes SatPipe to perform significantly worse than BBRv1 and LeoCC specifically during the reconfiguration phase (§B.2).

Phase 4: Post-Reconfiguration. Following the reconfiguration, we observe that performance is generally worse than in the pre-reconfiguration phase. The CCAs start probing for bandwidth from a much lower sending rate than after slow start, leading to a slow and less complete recovery. Again, Illinois's use of delay as a signal helps it slightly, but the explicit bandwidth probing of the model-based CCAs (BBRv1, BBRv3, LeoCC, SatPipe) proves most effective in this volatile environment.

Starlink's dynamic bandwidth allocation and proactive queue management create a challenging environment for traditional TCP congestion control. Model-based algorithms like BBR are more resilient to these dynamics than loss-based algorithms like CUBIC. To achieve optimal performance on Starlink, CCAs must tolerate RTT inflation during bandwidth probing and accept some packet losses to maintain their bandwidth allocation. Finally, a mechanism akin to slow start is needed after reconfigurations to explore the changed performance characteristics and reclaim bandwidth.

> *Takeaway* — Model-based CCAs (esp. BBRv1, LeoCC, SatPipe) outperform loss-based algorithms by tolerating AQM-induced losses. Slow start must sustain queue pressure to trigger demand-driven bandwidth allocation despite losses and rising RTT. Reconfigurations reset bandwidth allocation, requiring aggressive re-probing; LEO-specific mechanisms that reduce sending rate during handovers delay rather than accelerate this recovery.

5 Discussion & Conclusion

Starlink's sub-second dynamics differ substantially from terrestrial networks and from the assumptions embedded in most transport protocols. Our measurements uncover three interacting mechanisms that create an environment where conventional congestion control wisdom does not always apply. *First*, head-drop queuing with queue sizes of ≈ 1500 packets (DL) and ≈ 4000 packets (UL). *Second*, demand-driven bandwidth allocation that ramps up over ≈ 400 ms by factors of 3.4× (DL) and 2× (UL). *Third*, active queue management that aggressively controls queue occupancy, particularly on the uplink. These mechanisms interact with Starlink's 15-second reconfiguration cycle [23, 36], which resets bandwidth allocations and causes transient disruptions. We now discuss the implications of these findings.

Transport Protocol Design. Our findings reveal a fundamental tension in designing congestion control for Starlink. To claim additional bandwidth, a sender must build a queue. Yet doing so triggers AQM-induced losses and risks filling the shared DL buffer that couples loss across flows. Loss-based CCAs like CUBIC [26] interpret these AQM drops as congestion signals and reduce their sending rate, failing to sustain the pressure needed to maintain bandwidth allocation. Model-based algorithms like BBRv1 [6] and BBRv3 [7] prove more resilient because they probe for bandwidth independently of loss. This aligns with recent findings showing BBR outperforms CUBIC on Starlink [21].

The reconfiguration cycle compounds these challenges. Every 15 seconds, the bandwidth allocation resets, requiring CCAs to re-probe for capacity. Algorithms unaware of this periodicity waste time ramping back up, while handover-aware approaches like SatPipe [52], LeoCC [32], StarTCP [30], and SaTCP [5] show promise but require explicit knowledge of reconfiguration timing. Our examination of SatPipe highlights a related challenge: tracking the frequently shifting post-reconfiguration RTT. Because BBR relies on the minimum RTT to estimate the bandwidth-delay product and dictate the sending rate, accurate measurement is critical. Future algorithms must carefully weigh how probing for the minimum RTT on an idle link impacts Starlink's bandwidth allocation decisions. Slow start algorithms face a similar tension. They must apply enough pressure to trigger bandwidth allocation without filling the buffer so quickly that AQM induces heavy losses. HyStart++ [4] and similar conservative approaches exit slow start prematurely, failing to capitalize on Starlink's capacity, while vanilla exponential slow start overshoots and suffers significant packet loss.

Emulation and Simulation. Our characterization provides the foundation for more accurate modeling of LEO satellite networks. Existing network emulators like Mahimahi [37] typically assume tail-drop queues, static bandwidth, and loss patterns that do not capture Starlink's behavior. Recent efforts to build LEO-aware emulators [40, 46] could incorporate head-drop semantics, demand-driven bandwidth allocation, and periodic reconfiguration events. Without these features, protocol evaluations risk drawing conclusions that do not generalize to real deployments. The flow queuing behavior we uncover also has implications for fairness studies. Starlink's mechanism provides OWD isolation between flows, benefiting latency-sensitive applications sharing the link with bulk transfers. However, the shared DL queue couples loss rates across flows. This asymmetry means that fairness analyses must consider OWD and loss separately rather than assuming they correlate.

Limitations and Future Work. Several technical questions remain. Importantly, the exact AQM algorithm is not fully characterized. We find correlations with queue size and load-dependent behavior consistent with adaptive schemes like A-RED [17] or CoDel [39], but the precise mechanism remains proprietary. The interaction between bandwidth allocation and terminal density also warrants investigation. Finally, inter-satellite links (ISLs) may introduce additional dynamics not captured in our current measurements, as all our VPs followed a direct bent-pipe connection. Although ISLs are part of Starlink's architecture, prior work [36] indicates they are currently activated only in specific cases, and identifying whether a particular connection traverses an ISL remains non-trivial as this information is not exposed to end hosts. Characterizing how ISLs affect queuing behavior (queue sizing, AQM response, bandwidth allocation) is a natural next step, and the analysis techniques we presented are directly applicable.

Additionally, the behaviors highlighted in this paper reflect Starlink-specific engineering decisions. While one can argue that these choices are transient and may evolve as Starlink iterates on its design, they are currently the reality that application (and transport) protocols must contend with. As the first and largest commercial LEO operator, Starlink's design choices are likely to influence other satellite network operators now entering the market, such as OneWeb and Amazon's LEO. Our work demonstrates and provides a systematic methodology on investigating these behaviors at a microscopic level through controlled measurements and queue state estimation, which can be applied to other satellite networks as they become more accessible.

Acknowledgements

GenAI was used for grammar checking and coding assistance. This research was supported by the National Growth Fund through the Dutch 6G flagship project "Future Network Services".

References

[1] Ali Ahangarpour. 2024. *SatInView: Satellite Identification Tool.* Retrieved January 15, 2026 from https://github.com/aliahan/SatInView

[2] Ali Ahangarpour, Jinwei Zhao, and Jianping Pan. 2024. Trajectory-based Serving Satellite Identification with User Terminal's Field-of-View. In *Proceedings of the 2nd International Workshop on LEO Networking and Communication* (Washington, DC, USA) *(LEO-NET '24)*. Association for Computing Machinery, New York, NY, USA, 55–60. doi:10.1145/3697253.3697266

[3] Mahdi Arghavani, Haibo Zhang, David Eyers, and Abbas Arghavani. 2024. SUSS: Improving TCP Performance by Speeding Up Slow-Start. In *Proceedings of the ACM SIGCOMM 2024 Conference* (Sydney, NSW, Australia) *(ACM SIGCOMM '24)*. Association for Computing Machinery, New York, NY, USA, 151–165. doi:10.1145/3651890.3672234

[4] Praveen Balasubramanian, Yi Huang, and Matt Olson. 2023. HyStart++: Modified Slow Start for TCP. RFC 9406. doi:10.17487/RFC9406

[5] Xuyang Cao and Xinyu Zhang. 2023. SaTCP: Link-Layer Informed TCP Adaptation for Highly Dynamic LEO Satellite Networks. In *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*. IEEE, Piscataway, NJ, USA, 1–10. doi:10.1109/INFOCOM53939.2023.10228914

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

[7] Neal Cardwell, Ian Swett, and Joseph Beshay. 2025. *BBR Congestion Control*. Internet-Draft draft-ietf-ccwg-bbr-04. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-ccwg-bbr/04/ Work in Progress.

[8] Hendrik Cech. 2026. *Dissecting the StarLink: Scripts.* Retrieved June 15, 2026 from https://github.com/hendrikcech/dissecting-starlink

[9] Hendrik Cech. 2026. *NetScalpel.* Retrieved June 15, 2026 from https://github.com/hendrikcech/NetScalpel

[10] Hendrik Cech. 2026. *pcap-match.* Retrieved June 15, 2026 from https://github.com/hendrikcech/pcap-match

[11] Hendrik Cech, Nitinder Mohan, and Jörg Ott. 2026. *Dissecting the StarLink: Characterizing Queuing and Flow Dynamics in the Starlink Network.* doi:10.14459/2026mp1856124

[12] Chen Chen, Pavel Chikulaev, Sergii Ziuzin, David Sacks, Peter J. Worters, Darshan Purohit, Yashodhan Dandekar, Vladimir Skuratovich, Andrei Pushkin, and Phillip E. Barber. 2024. Low latency schedule-driven handovers. Retrieved October 20, 2025 from https://patents.google.com/patent/US11949496B1 Patent No. 11949496B1, Issued Apr. 2024.

[13] Jae Won Chung, Maryam Ataei Kachooei, Feng Li, and Mark Claypool. 2025. *SEARCH – a New Slow Start Algorithm for TCP and QUIC*. Internet-Draft draft-chung-ccwg-search-07. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-chung-ccwg-search/07/ Work in Progress.

[14] Amazon Elastic Compute Cloud. 2024. *Precision clock and time synchronization on your EC2 instance.* Retrieved January 30, 2026 from https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/set-time.html

[15] Ding Zhao. 2025. *SatPipe Implementation.* Retrieved May 21, 2026 from https://github.com/dzhao99/SatPipe

[16] Saeed Fadaei, Shubham Tiwari, Aryan Taneja, Saksham Bhushan, Mohamed Kassem, Aravindh Raman, Debopam Bhattacherjee, Lili Qiu, Alan Woodward, and Nishanth Sastry. 2025. LEOScope: Building a Global Testbed for Low-Earth Orbit Satellite Networks. *SIGCOMM Comput. Commun. Rev.* 55, 2 (July 2025), 13–21. doi:10.1145/3750832.3750835

[17] Sally Floyd, Ramakrishna Gummadi, Scott Shenker, et al. 2001. Adaptive RED: An algorithm for increasing the robustness of RED's active queue management.

[18] S. Floyd and V. Jacobson. 1993. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking* 1, 4 (1993), 397–413. doi:10.1109/90.251892

[19] Johan Garcia, Simon Sundberg, and Anna Brunstrom. 2024. Inferring Starlink Physical Layer Transmission Rates Through Receiver Packet Timestamps. In *2024 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, Piscataway, NJ, USA, 1–6. doi:10.1109/WCNC57260.2024.10570978

[20] Johan Garcia, Simon Sundberg, and Anna Brunstrom. 2025. A Detailed Characterization of Starlink One-way Delay. In *Proceedings of the 2025 3rd Workshop on LEO Networking and Communication* (Coimbra, Portugal) *(LEO-NET '25)*. Association for Computing Machinery, New York, NY, USA, 43–49. doi:10.1145/3748749.3749090

[21] Johan Garcia, Simon Sundberg, and Anna Brunstrom. 2025. TCP Congestion Control Performance over Starlink. In *Proceedings of the 2025 Applied Networking Research Workshop* (Madrid, Spain) *(ANRW '25)*. Association for Computing Machinery, New York, NY, USA, 70–77. doi:10.1145/3744200.3744760

[22] Johan Garcia, Simon Sundberg, and Anna Brunstrom. 2026. Characterizing the Configuration of Starlink Queuing. In *Proceedings of the 2026 ACM Internet Measurement Conference* (Karlsruhe, Germany) *(IMC '26)*. Association for Computing Machinery, New York, NY, USA, 10 pages. doi:10.1145/3777912.3809153

[23] Johan Garcia, Simon Sundberg, Giuseppe Caso, and Anna Brunstrom. 2023. Multi-Timescale Evaluation of Starlink Throughput. In *Proceedings of the 1st ACM Workshop on LEO Networking and Communication* (Madrid, Spain) *(LEO-NET '23)*. Association for Computing Machinery, New York, NY, USA, 31–36. doi:10.1145/3614204.3616108

[24] Jeffrey A. Gardner, Yashodhan Dandekar, Ramakrishna Akella, and Khasim Shaheed Shaik Mahammad. 2025. System and method of providing a radio link control sublayer. Retrieved October 20, 2025 from https://patents.google.com/patent/US20250125905A1/en Patent No. 20250125905A1, Issued Apr. 2025.

[25] Google. 2024. *BBR Congestion Control Implementation.* Retrieved January 15, 2026 from https://github.com/google/bbr

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

[27] Todd E. Humphreys, Peter A. Iannucci, Zacharias M. Komodromos, and Andrew M. Graff. 2023. Signal Structure of the Starlink Ku-Band Downlink. *IEEE Trans. Aerospace Electron. Systems* 59, 5 (2023), 6016–6030. doi:10.1109/TAES.2023.3268610

[28] Jayasuryan V. Iyer, Khasim Shaheed Shaik Mahammad, Yashodhan Dandekar, Ramakrishna Akella, Chen Chen, Phillip E. Barber, and Peter J. Worters. 2023. System and method of providing a medium access control scheduler. Retrieved October 20, 2025 from https://patents.google.com/patent/US11800552B1 Patent No. 11800552B1, Issued Oct. 2023.

[29] J. Jaffe. 1981. Bottleneck Flow Control. *IEEE Transactions on Communications* 29, 7 (1981), 954–962. doi:10.1109/TCOM.1981.1095081

[30] Li Jiang, Yihang Zhang, Yannan Hu, Yong Cui, and Xinggong Zhang. 2024. StarTCP: Handover-aware Transport Protocol for Starlink. In *Proceedings of the 8th Asia-Pacific Workshop on Networking* (Sydney, Australia) *(APNet '24)*. Association for Computing Machinery, New York, NY, USA, 169–170. doi:10.1145/3663408.3665803

[31] Victor Kamel, Jinwei Zhao, Daoping Li, and Jianping Pan. 2024. StarQUIC: Tuning Congestion Control Algorithms for QUIC over LEO Satellite Networks. In *Proceedings of the 2nd International Workshop on LEO Networking and Communication* (Washington, DC, USA) *(LEO-NET '24)*. Association for Computing Machinery, New York, NY, USA, 43–48. doi:10.1145/3697253.3697271

[32] Zeqi Lai, Zonglun Li, Qian Wu, Hewu Li, Jihao Li, Xin Xie, Yuanjie Li, Jun Liu, and Jianping Wu. 2025. LeoCC: Making Internet Congestion Control Robust to LEO Satellite Dynamics. In *Proceedings of the ACM SIGCOMM 2025 Conference* (São Francisco Convent, Coimbra, Portugal) *(SIGCOMM '25)*. Association for Computing Machinery, New York, NY, USA, 129–146. doi:10.1145/3718958.3750491

[33] Shao Liu, Tamer Başar, and R. Srikant. 2006. TCP-Illinois: a loss and delay-based congestion control algorithm for high-speed networks. In *Proceedings of the 1st International Conference on Performance Evaluation Methodolgies and Tools* (Pisa, Italy) *(valuetools '06)*. Association for Computing Machinery, New York, NY, USA, 55–es. doi:10.1145/1190095.1190166

[34] Zikun Liu, Fan-Xue Gabriella Reidys, Sarah Tanveer, and Deepak Vasisht. 2025. Vivisecting Starlink Throughput: Measurement and Prediction. *Proc. ACM Netw.* 3, CoNEXT4, Article 24 (Nov. 2025), 23 pages. doi:10.1145/3768971

[35] François Michel, Martino Trevisan, Danilo Giordano, and Olivier Bonaventure. 2022. A first look at starlink performance. In *Proceedings of the 22nd ACM Internet Measurement Conference* (Nice, France) *(IMC '22)*. Association for Computing Machinery, New York, NY, USA, 130–136. doi:10.1145/3517745.3561416

[36] Nitinder Mohan, Andrew E. Ferguson, Hendrik Cech, Rohan Bose, Prakita Rayyan Renatin, Mahesh K. Marina, and Jörg Ott. 2024. A Multifaceted Look at Starlink Performance. In *Proceedings of the ACM Web Conference 2024* (Singapore, Singapore) *(WWW '24)*. Association for Computing Machinery, New York, NY, USA, 2723–2734. doi:10.1145/3589334.3645328

[37] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. 2015. Mahimahi: Accurate Record-and-Replay for HTTP. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. USENIX Association, Santa Clara, CA, 417–429. https://www.usenix.org/conference/atc15/technical-session/presentation/netravali

[38] Kathleen Nichols and Van Jacobson. 2012. Controlling queue delay. *Commun. ACM* 55, 7 (2012), 42–50.

[39] Kathleen Nichols, Van Jacobson, Andrew McGregor, and Jana Iyengar. 2018. Controlled Delay Active Queue Management. RFC 8289. doi:10.17487/RFC8289

[40] Robin Ohs, Gregory F. Stock, Juan A. Fraire, Holger Hermanns, and Andreas Schmidt. 2025. PhantomLink: Emulating Virtual End-to-End Links on Ground and in Orbit. In *Proceedings of the 2025 Applied Networking Research Workshop* (Madrid, Spain) *(ANRW '25)*. Association for Computing Machinery, New York, NY, USA, 39–46. doi:10.1145/3744200.3744758

[41] Rong Pan, Preethi Natarajan, Fred Baker, and Greg White. 2017. Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem. RFC 8033. doi:10.17487/RFC8033

[42] Rong Pan, Preethi Natarajan, Chiara Piglione, Mythili Suryanarayana Prabhu, Vijay Subramanian, Fred Baker, and Bill VerSteeg. 2013. PIE: A lightweight control scheme to address the bufferbloat problem. In *2013 IEEE 14th International Conference on High Performance Switching and Routing (HPSR)*. IEEE, Piscataway, NJ, USA, 148–155. doi:10.1109/HPSR.2013.6602305

[43] PeeringDB. 2026. *SpaceX Starlink.* Retrieved June 3, 2026 from https://www.peeringdb.com/net/18747
[44] Project-Faster. 2024. *TCP Search Congestion Control Implementation.* Retrieved January 15, 2026 from https://github.com/Project-Faster/tcpss_search
[45] Jeanine Romano, Jeffrey D Kromrey, Jesse Coraggio, and Jeff Skowronek. 2006. Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys. In *annual meeting of the Florida Association of Institutional Research*, Vol. 177. New York, NY, USA, 1–10. Available at https://www.scribd.com/document/45715000/Appropriate-Statistics-for-Ordinal-Level-Data.
[46] William Sentosa, Balakrishnan Chandrasekaran, P. Brighten Godfrey, and Haitham Hassanieh. 2025. CellReplay: Towards accurate record-and-replay for cellular networks. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. USENIX Association, Philadelphia, PA, 1169–1186. https://www.usenix.org/conference/nsdi25/presentation/sentosa
[47] SpaceNetLab. 2024. *LeoCC Implementation.* Retrieved January 15, 2026 from https://github.com/SpaceNetLab/LeoCC
[48] SpaceX. 2026. *FORM S-1, Space Exploration Technologies Corp.* Retrieved May 21, 2026 from https://www.sec.gov/Archives/edgar/data/1181412/000162828026036936/spaceexplorationtechnologi.htm
[49] SUSSdeveloper. 2024. *HyStart++ Implementation.* Retrieved January 15, 2026 from https://github.com/SUSSdeveloper/HyStartPP
[50] SUSSdeveloper. 2024. *SUSSprg: SUSS Congestion Control Implementation.* Retrieved January 15, 2026 from https://github.com/SUSSdeveloper/SUSSprg
[51] Hammas Bin Tanveer, Mike Puchol, Rachee Singh, Antonio Bianchi, and Rishab Nithyanand. 2023. Making Sense of Constellations: Methodologies for Understanding Starlink's Scheduling Algorithms. In *Companion of the 19th International Conference on Emerging Networking EXperiments and Technologies* (Paris, France) *(CoNEXT 2023)*. Association for Computing Machinery, New York, NY, USA, 37–43. doi:10.1145/3624354.3630586
[52] Ding Zhao, Xinyu Zhang, and Myungjin Lee. 2025. SatPipe: Deterministic TCP Adaptation for Highly Dynamic LEO Satellite Networks. In *IEEE INFOCOM 2025 - IEEE Conference on Computer Communications*. IEEE, Piscataway, NJ, USA, 1–10. doi:10.1109/INFOCOM55648.2025.11044600

A Appendix

Appendices are supporting material that has not been peer-reviewed.

A.1 SwitchFlow Experiment

The `SwitchFlow` experiment collects data to analyze whether bandwidth allocation is per-UT or per-flow. The result is discussed in §4.2. Flow A is active for 800 ms (at a rate varied between experiments) before flow B takes over immediately, which continues sending for another 800 ms. The total receive rate of flow A and B is plotted (black line); it does not show a decrease at the switchpoint, indicating that flow B can immediately fall back on flow A's bandwidth allocation. At higher send rates, flow B's individual receive rate progressively increases for ≈ 200–400 ms. During this time, flow B still occupies space in the queue and parts of the available bandwidth.

A.2 AQM Sojourn Time Analysis

We analyse the correlation between queue size and frame loss frequency in §4.3. Figure 14 shows the correlation of sojourn time and frame loss frequency. The frame loss frequency on the UL increases along with sojourn time, similar to queue size. However, the gradient differs between workloads, indicating that the mechanism does not strictly react to sojourn time. On the DL, we observe loss frequency spikes at different sojourn times; these correlate with packets hitting the queue size limit of 1500 packets. Due to the demand-driven bandwidth allocation, lower send rates elicit less bandwidth, which leads to higher sojourn times at the same queue size.

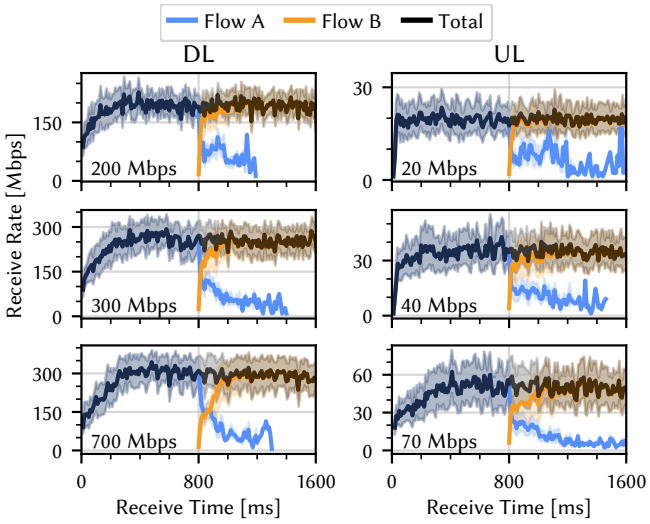
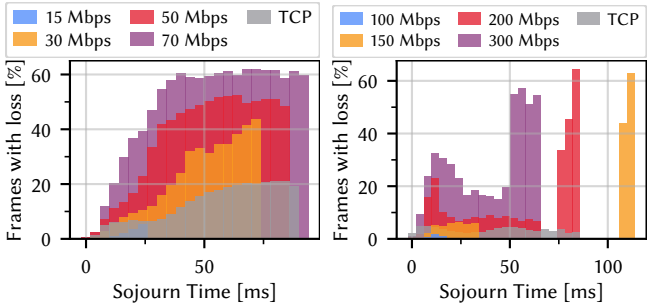


Figure 13: Flow A experiences the demand-driven bandwidth allocation from 0–400 ms. It stops sending after 800 ms when flow B starts. The total receive rate stays constant, indicating that the bandwidth is allocated per-UT instead of per-flow (see §4.2).



(a) Uplink (UL). **(b) Downlink (DL).**

Figure 14: The sojourn time correlates with loss frequency.

A.3 MouseEleph Experiment

In §4.5, we analyze the behavior of two concurrent flows. The first *elephant* flow sends at a fixed rate of 250/140 Mbps on DL/UL, respectively, while the send rate of the parallel *mouse* flow is varied across runs. Figure 10 shows the OWD and loss rates of this experiment. Figure 15 plots the estimated queue occupancy of the flows. We observe that the sum of the DL queues "above fair share" at the send rates of 50, 100, and 150 Mbps sum up to 1500 packets, which is the DL queue size inferred in §4.1.

A.4 Cooldown Experiment

In §4.2, we analyze the decay rate of the bandwidth allocation if sending stops. Figure 6 visualizes a test with send rates of 500 Mbps on the DL and 140 Mbps on the UL. We repeated the test with a lower UL send rate of 70 Mbps and plot the results in Figure 16. We

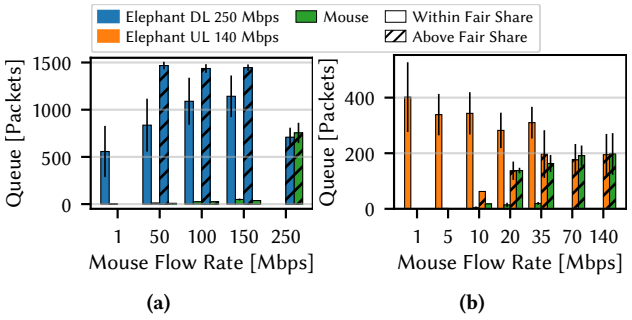


Figure 15: Due to a shared queue, not more than ≈ 1500 packets can be stored in total on the DL (a). On the UL (b), the AQM mechanism keeps the queue sizes low and prevents the coupled loss behavior seen on the DL.

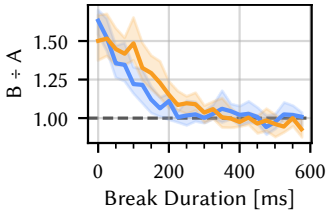


Figure 16: The cooldown experiment repeated with a lower UL send rate of 70 Mbps. The bandwidth allocation decays more quickly compared to the higher send rate in Figure 6.

observe that the allocation plateau, present in the 140 Mbps test, disappears when the experiment is repeated at the lower send rate of 70 Mbps. The lower send rate fills the queue less, suggesting that the scheduler also considers queue occupancy when deciding how long to maintain an allocation.

A.5 Reconfiguration Loss Rate

Figure 17 shows the loss rate of the saturating UDP transfers from §4.4 around the reconfiguration point. Note that steady-state loss is expected since the send rates (500/200 Mbps on DL/UL) exceed the link capacity. On the DL, the mean loss rate reaches 100% during the connection break because the queue is already at capacity during normal operation and cannot buffer additional packets. On the UL, the loss rate spikes but does not reach 100%, confirming that the deep UL queue is able to buffer packets during the break. This is possible because the aggressive AQM scheme keeps the queue utilization low during steady state, leaving headroom to absorb the burst.

A.6 Idle Link Metrics

Figure 18 shows data from the OWD experiment, which sends a UDP probe packet once per millisecond to characterize the mostly-idle link. Both OWD and loss rate are stable within reconfiguration intervals but spike at the reconfiguration point. The average base OWD is 13 ms on the DL and 17 ms on the UL. The average stable loss rates are 0.03% on the DL and 0.02% on the UL.

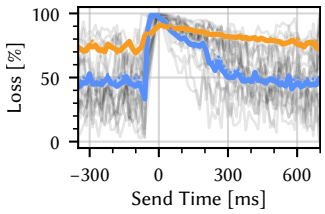


Figure 17: Loss rate around the reconfiguration point on DL (blue) and UL (orange). Solid lines indicate a satellite handover; dashed lines indicate no handover.

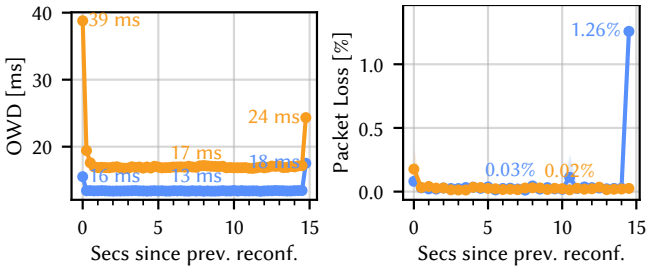


Figure 18: Mostly-idle link performance measured with 1 ms UDP probes on DL (blue) and UL (orange).

B Significance Tests

We substantiate the central claims of our analysis with statistical significance tests. Throughout, we employ non-parametric tests, which do not assume normally distributed data. We reject null hypotheses at a significance level of $p < 0.05$. Significance levels are denoted as: $^{*}: p < 0.05$; $^{**}: p < 0.01$; $^{***}: p < 0.001$. To quantify the magnitude of observed differences independent of sample size, we report effect sizes alongside each test.

B.1 Flow Queuing Significance Tests

We substantiate the central claims of our flow queuing analysis in §4.5 with significance tests. First, we test the hypothesis that both mouse and elephant flows "experience nearly identical loss rates on the DL". We compare the loss rate differences per mouse-rate in Table 1 and test the null hypothesis that the differences are zero with a paired Wilcoxon signed-rank test. We pick a paired test since the elephant and mouse tests are not independent; they are running at the same time over the same link. Table 1 shows that the null hypothesis is rejected for all mouse rates smaller than the elephant rate. It also confirms that the elephant loss rates are consistently up to 2 pp higher than the mouse loss rates, likely caused by the elephant flow contributing proportionally more packets to the shared queue. To formally evaluate our claim of "nearly identical" loss rates, we conduct a Two One-Sided Tests (TOST) procedure for equivalence. We choose an equivalence margin of $\Delta = \pm 2$ pp based on the maximum observed mean difference. The TOST is highly significant ($p < 10^{-17}$), confirming that mouse and elephant flows share statistically equivalent (±2 pp) loss rates on the DL.

Table 1: Downlink (DL) paired Wilcoxon signed-rank significance test. Elephant send rate was 250 Mbps.

Rate [Mbps]	n	Effect Size (r)	Significance	Mean Eleph. [%]	Mean Mouse [%]	Mean Difference [%]
1	65	+0.44	**	9.13	8.47	0.66
50	50	+0.70	***	19.67	17.77	1.90
100	59	+0.91	***	21.72	20.12	1.59
150	55	+0.92	***	28.54	26.73	1.81
250	71	+0.09	n.s.	43.98	43.99	-0.01

Table 2: Uplink (UL) paired Wilcoxon signed-rank significance test with measurements where the mouse flow stayed within its fair share of link capacity. Elephant send rate was 140 Mbps.

Rate [Mbps]	n	Effect Size (r)	Significance	Mean Eleph. [%]	Mean Mouse [%]	Mean Difference [%]
1	81	+1.00	***	61.95	0.20	61.75
5	71	+1.00	***	65.88	0.24	65.64
10	77	+1.00	***	66.09	0.48	65.61
20	8	+1.00	**	67.08	0.56	66.51
35	23	+1.00	***	62.60	0.59	62.00
70	0					
140	0					

Second, we substantiate our claim that the elephant and mouse loss rates are not coupled on the UL if the mouse rate stays within its fair share of the link capacity. We again conduct a paired Wilcoxon signed-rank test per mouse-rate and show the results in Table 2. The test is significant for all tested sub-fair-share mouse rates; the rank-biserial effect size of $r = 1.0$ shows perfect rank separation, meaning the elephant loss exceeded the mouse loss rate in every single test run.

B.2 TCP Significance Tests

The TCP performance tests that we ran on Starlink are shown and discussed in §4.6. To further strengthen our analysis, we compare the performance of the CCAs within each phase (slow start, pre-reconfiguration, reconfiguration, and post-reconfiguration) pairwise. For that purpose, we compare the mean goodput within each phase. To test the null hypothesis that the goodput distributions of two CCAs are identical, we use the two-sided Mann-Whitney U test. We chose this test as it is designed for comparing independent samples (since the CCA measurements are run sequentially at different times). As effect size, we compute Cliff's delta ($\delta \in [-1, 1]$). A value of 1 (or -1) indicates that every observation for CCA A is strictly greater (or smaller) than every observation for CCA B, whereas 0 indicates completely overlapping distributions. We label the magnitude of Cliff's delta according to the common classification of Romano et al. [45]: < 0.147 negligible, 0.147–0.33 small, 0.33–0.474 medium, > 0.474 large.

Figure 19 visualizes the results, split into phase and link direction. A positive δ (red colors) means the row CCA achieves higher goodput than the column CCA. A negative δ (blue colors) means the column CCA's goodput is higher. Empty cells indicate $p \geq 0.05$.

Our tests confirm the clear separation of algorithms into distinct performance groups. On the upper end, the model-based CCAs BBRv1, LeoCC, and SatPipe perform best throughout all phases (including slow-start). SatPipe, however, experiences a significant performance drop during the reconfiguration phase. On the lower end is the loss-based CCA CUBIC (along with its slow-start variants). BBRv3 and Illinois fall between these two performance poles. BBRv3 performs significantly worse than the pure model-based CCAs, while Illinois achieves substantially higher goodput than the other loss-based CCAs. Finally, the pairwise analysis of the slow-start phase confirms our earlier observations, highlighting in particular the poor performance of HyStart, HyStart++, and SUSS.

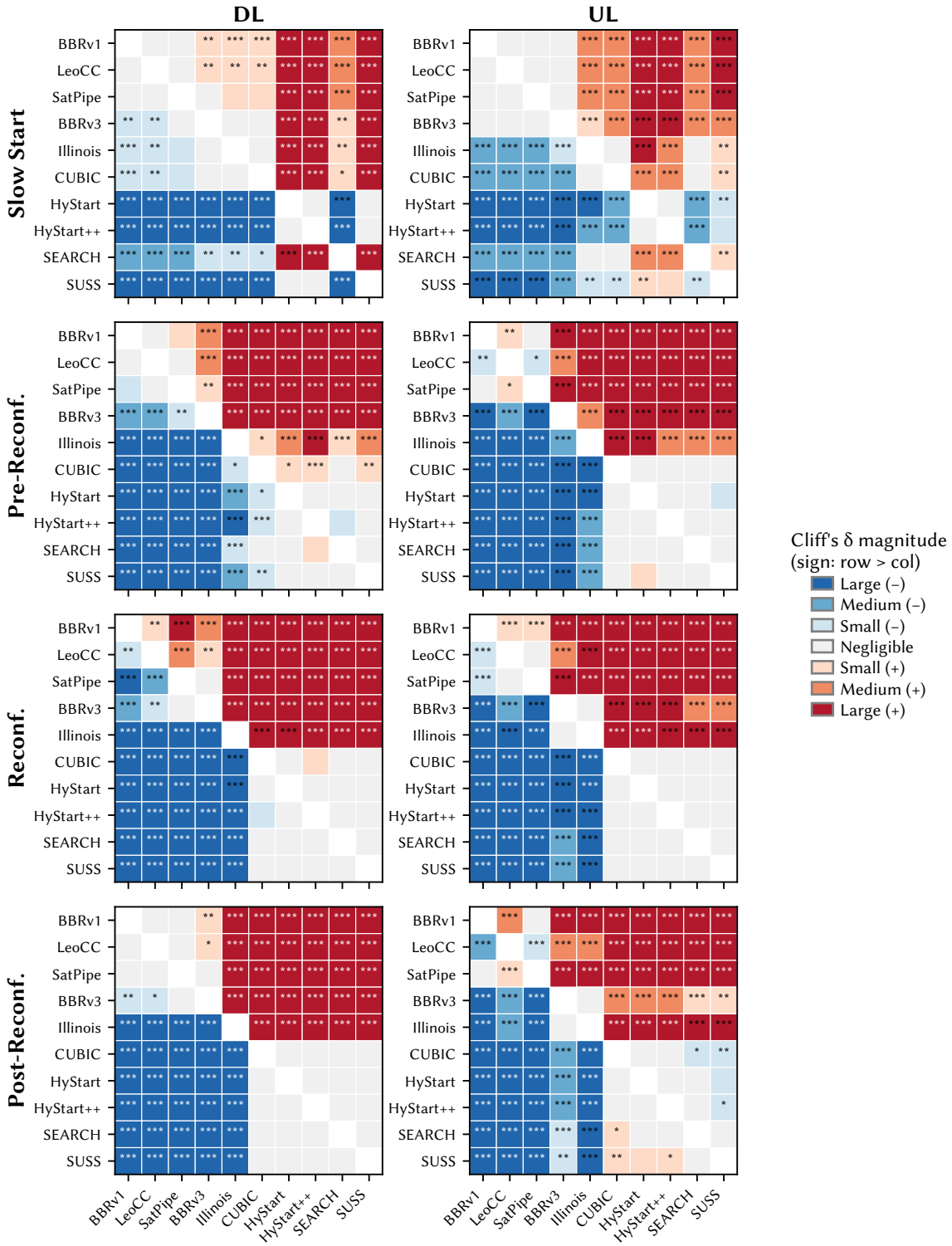


Figure 19: The pairwise significance and effect size analysis of our TCP measurements, split by phase and link direction. The cell colors are based on the classified Cliff's delta value. The analysis is discussed in §B.2; the results are first shown in §4.6.

C Measurement Data

Table 3 summarizes the final dataset analyzed in this paper, listing all experiment configurations, their parameters, and the number of repetitions per vantage point. The primary campaign was conducted from Munich, with key experiments replicated from Victoria and Waterloo to verify generalizability across locations and constellations. Parameters were randomized across runs to avoid interference from measurement order. The column *Min. Samples* reports the minimum number of repetitions for any single parameter configuration, while *Total* gives the overall number of runs.

Table 3: The final dataset depicted and analyzed for this paper. All data is available in [11]. Column *Total* contains the total number of experiment repetitions, i.e., how often the experiment was conducted with a random choice of parameters from the listed parameters. Column *Min. Samples* shows the minimum number of repetitions of each experiment configuration. Since the parameters of the next experiment were chosen randomly (to avoid interference from measurement order), the measurement configurations were not picked the same number of times.

Location	Experiment		Min. Samples	Total
Munich	`Cooldown`	$rateUL = 140$, $rateDL = 500$	106	6,011
	`Cooldown`	$rateUL = 70$, $rateDL = 300$	94	5,688
	`MouseElephant`	$mouseRatesUL = \{1, 5, 10, 20, 35, 70, 140\}$ $mouseRatesDL = \{1, 50, 100, 150, 250\}$	50	1,642
	`MultiDurationRate`	$duration = 1$s $ratesUL = \{5, 10, 15, 20, 25, 30, 35, 40, 50, 60, 70, 80, 90, 100, 110, 140\}$ $ratesDL = \{50, 75, 100, 125, 150, 175, 200, 250, 300, 350, 400, 500, 700\}$	70	5,376
	`MultiDurationRate`	$duration = 4$s $ratesUL = \{15, 30, 50, 70, 90, 140, 200, 500\}$ $ratesDL = \{50, 100, 150, 200, 300, 400, 500, 600, 700\}$	103	2,100
	`OWD`	$duration = 60$s	130	522
	`Rate`	$duration = 60$s	63	131
	`SwitchFlow`	$ratesUL = \{200, 300, 700\}$ $ratesDL = \{200, 250, 300, 700\}$	35	299
	`TCPReconf`	$direction = 12$s $ccas$ = { UDP, BBRv1, BBRv3, LeoCC, SatPipe, Illinois, CUBIC, HyStart, HyStart++, SEARCH, SUSS }	77	2,057
	`Burst`	$packetPadding = \{0, 700, 1400\}$ $packetCount = \{1000, 3000, 6000, 10000\}$	68	1,849
Victoria	`Burst`	$packetPadding = \{0, 700, 1400\}$ $packetCountUL = \{10000\}$ $packetCountDL = \{3000, 6000, 10000\}$	UL = 60 DL = 20	360
	`MouseElephant`	$mouseRatesUL = \{1, 10, 35, 140\}$ $mouseRatesDL = \{1, 50, 100, 250\}$	UL = 85 DL = 53	480
	`Cooldown`	$rateUL = 140$, $rateDL = 500$, $step = 50$	100	2,941
	`Rate`	$duration = 60$s	34	69
Waterloo	`MultiDurationRate`	$duration = 1$s $ratesUL = \{35, 70, 140\}$ $ratesDL = \{100, 300, 500, 700\}$	361	2,545
	`OWD`	$duration = 60$s	32	72

D `NetScalpel` Experiments

Table 4 details the scheduling algorithms of the eight experiments introduced in §3.1. Each algorithm composes the measurement primitives provided by `NetScalpel` (fixed-rate UDP, periodic probes, bursts, and TCP transfers) into a test schedule for one reconfiguration interval.

Table 4: Experiment Algorithms supported by `NetScalpel`. *schedule* ← **adds a new test to the schedule. The variable** *nextReconf* **holds the timestamp of the next 15 s reconfiguration point. Rates are specified in Mbps.**

Experiment & Notes	Algorithm Logic

`MultiDurationRate`
 	
Schedule fixed-rate UDP tests with variable rates and durations within a reconfiguration interval.
	Require: $direction \in \{\text{DL}, \text{UL}\}$
	Require: $ratesUL$ (default $\{15, 30, 50, 70, 90, 140, 200\}$)
	Require: $ratesDL$ (default $\{50, 100, 150, 200, 300, 400, 500, 600, 700\}$)
	Require: $durations$ (default $\{1000, 4000\}$)
	$start \leftarrow nextReconf + 1\,\text{s}$
	$deadline \leftarrow nextReconf + 14\,\text{s}$
	$rate \leftarrow (direction == \text{UL}\ ?\ ratesUL : ratesDL)$
	for all $rate \in \textsc{Shuffle}(rates)$ **do**
	  for all $duration \in \textsc{Shuffle}(durations)$ **do**
	  if $start + duration > deadline$ **then**
	  return
	  end if
	  $schedule \leftarrow \textsc{FixedRateUDP}(direction, start, rate, duration)$
	  $schedule \leftarrow \textsc{PeriodicUDP}(direction, start, 1000\,\text{Hz})$
	  $start \leftarrow start + duration + 2\,\text{s}$
	  end for
	end for

`Burst`
 	
Send packet bursts with varying size (*packetPadding*) and number of packets (*packetCount*).
	Require: $direction \in \{\text{DL}, \text{UL}\}$
	Require: $packetPadding$ (default $\{0, 700, 1400\}$)
	Require: $packetCount$ (default $\{3000, 10000\}$)
	$start \leftarrow nextReconf + 1\,\text{s}$
	$deadline \leftarrow nextReconf + 14\,\text{s}$
	for all $pad \in \textsc{Shuffle}(packetPadding)$ **do**
	  for all $num \in \textsc{Shuffle}(packetCount)$ **do**
	  if $start + 4\,\text{s} > deadline$ **then**
	  return
	  end if
	  $schedule \leftarrow \textsc{BurstUDP}(direction, start, num, pad)$
	  $start \leftarrow start + duration + 4\,\text{s}$
	  end for
	end for

Table 4: Experiment Algorithms supported by `NetScalpel` (Continued).

Experiment & Notes	Algorithm Logic

`Cooldown`
 	
Collects data for the bandwidth allocation decay analysis.
	Require: $direction \in \{\text{DL, UL}\}$
	Require: $rateUL$ (default 140)
	Require: $rateDL$ (default 500)
	$start \leftarrow nextReconf + 1\,\text{s}$
	$deadline \leftarrow nextReconf + 14\,\text{s}$
	$duration \leftarrow 800\,\text{ms}$
	$rate \leftarrow (direction == \text{UL} \; ? \; rateUL : rateDL)$
	$step \leftarrow 25$
	while true **do**
	  $break \leftarrow \textsc{Random}(0, 600, step)$
	  if $start + duration + break + duration > deadline$ **then**
	  return
	  end if
	  $\textsc{FixedRateUDP}(direction, start, rate, duration)$
	  $\textsc{FixedRateUDP}(direction, start + duration, 0, break)$
	  $\textsc{FixedRateUDP}(direction, start + duration + break, rate, duration)$
	  $start \leftarrow start + duration + break + 2\,\text{s}$
	end while

`TCPReconf`
 	
Measures TCP performance while it experiences a reconfiguration. Additionally, UDP is sometimes tested to measure a baseline performance.
	Require: $direction \in \{\text{DL, UL}\}$
	Require: $ccas$ (List of CCAs)
	Require: $duration$ (default 12 s)
	$start \leftarrow nextReconf - (duration/2)$
	$cca \leftarrow \textsc{Shuffle}(ccas)[0]$
	if $cca ==$ UDP **then**
	  $rate \leftarrow (direction == \text{UL} \; ? \; 200 : 500)$
	  $\textsc{FixedRateUDP}(direction, start, rate, duration)$
	else
	  $\text{TCP}(direction, start, cca, duration)$
	end if

`Rate`
 	
Long-duration throughput measurement.
	Require: $direction \in \{\text{DL, UL, bidir}\}$
	Require: $duration$ (default 60 s)
	$start \leftarrow nextReconf + 5\,\text{s}$
	if $direction \in \{\text{DL, UL, bidir}\}$ **then**
	  $\textsc{FixedRateUDP}(\text{UL}, start, 200, duration)$
	  $\textsc{PeriodicUDP}(\text{UL}, start, 1000\,\text{Hz})$
	end if
	if $direction \in \{\text{DL, bidir}\}$ **then**
	  $\textsc{FixedRateUDP}(\text{DL}, start, 500, duration)$
	  $\textsc{PeriodicUDP}(\text{DL}, start, 1000\,\text{Hz})$
	end if

`OWD`
 	
One-Way Delay (OWD) measurement without load.
	Require: $direction \in \{\text{DL, UL, bidir}\}$
	Require: $duration$ (default 60 s)
	$start \leftarrow nextReconf + 5\,\text{s}$
	if $direction \in \{\text{DL, UL, bidir}\}$ **then**
	  $\textsc{PeriodicUDP}(\text{UL}, start, 1000\,\text{Hz})$
	end if
	if $direction \in \{\text{DL, bidir}\}$ **then**
	  $\textsc{PeriodicUDP}(\text{DL}, start, 1000\,\text{Hz})$
	end if

Table 4: Experiment Algorithms supported by `NetScalpel` (Continued).

Experiment & Notes	Algorithm Logic

`MouseElephant`
 	
Collects data for the flow queueing analysis.
	Require: $direction \in \{\text{DL, UL}\}$
	Require: $elephRateDL$ (default 250)
	Require: $elephRateUL$ (default 140)
	Require: $mouseRatesDL$ (default $\{1, 50, 100, 150, 250\}$)
	Require: $mouseRatesUL$ (default $\{1, 5, 10, 20, 35, 70, 140\}$)
	$start \leftarrow nextReconf + 1\,\text{s}$
	$deadline \leftarrow nextReconf + 14\,\text{s}$
	$elephRate \leftarrow (direction == \text{UL} \ ? \ elephRate : elephRate)$
	$mouseRates \leftarrow (direction == \text{UL} \ ? \ mouseRatesUL : mouseRatesDL)$
	for all $mouseRate \in \textsc{Shuffle}(mouseRates)$ **do**
	  if $start + duration > deadline$ **then**
	  return
	  end if
	  $\textsc{FixedRateUDP}(\text{UL}, start, elephRate, 4\,\text{s})$
	  $\textsc{FixedRateUDP}(\text{UL}, start, mouseRate, 4\,\text{s})$
	  $start \leftarrow start + duration + 3\,\text{s}$
	end for

`SwitchFlow`
 	
Collects data for the analysis of bandwidth allocation behavior.
	Require: $direction \in \{\text{DL, UL}\}$
	Require: $ratesUL$ (default $\{20, 40, 70\}$)
	Require: $ratesDL$ (default $\{200, 300, 700\}$)
	$start \leftarrow nextReconf + 1\,\text{s}$
	$deadline \leftarrow nextReconf + 14\,\text{s}$
	$rates \leftarrow (direction == \text{UL} \ ? \ ratesUL : ratesDL)$
	$duration \leftarrow 800\,\text{ms}$
	for all $rate \in \textsc{Shuffle}(rates)$ **do**
	  if $start + (2 \cdot duration) > deadline$ **then**
	  return
	  end if
	  $schedule \leftarrow \textsc{FixedRateUDP}(direction, start, rate, duration)$
	  $schedule \leftarrow \textsc{FixedRateUDP}(direction, start + duration, rate, duration)$
	  $start \leftarrow start + (2 \cdot duration) + 2\,\text{s}$
	end for