

Disentangling Reordering, Packet Loss and Transport Performance in LEO Satellite Networks

Minjun Choi, Mingyu Park, and Jeongyeup Paek

Abstract—Low Earth Orbit (LEO) satellite networks such as Starlink provide broadband Internet connectivity in areas where terrestrial infrastructure is limited. However, their fast movement and frequent path reconfigurations make the access link highly dynamic. This dynamism can introduce link-induced packet reordering and loss, both of which interfere with TCP's loss inference and congestion control. In this paper, we analyze the behavior of a commercial Starlink to quantify how reordering and packet loss arise and how they affect widely deployed TCP congestion-control algorithms. Through extensive experiments, we find that reordering occurs in both directions, but through different mechanisms. Uplink reordering is loss-free and sub-millisecond, suggesting scheduling or buffering, whereas downlink reordering comes with loss and takes milliseconds to resolve. We further demonstrate that loss is governed by the packet rate rather than the bit rate alone, and that a single per-packet service-time model reproduces the capacity of both links. On the downlink, small packets meet a hard packet-rate cap ($\approx 112,000$ pps) well below the bit-rate ceiling (≈ 296 Mbps). The uplink, in contrast, is bound by an ≈ 18 Mbps token-bucket policer. Finally, TCP experiments comparing several loss-detection mechanisms reveal that reordering-aware loss detection reduces retransmissions by 13–52%. The results indicate that transport performance over LEO networks is limited not only by capacity and delay variation, but also by reordering and packet-rate effects that should be considered in satellite-aware congestion control.

Index Terms—Starlink, LEO satellite networks, packet reordering, TCP, congestion control, network measurement

I. INTRODUCTION

Low Earth Orbit (LEO) satellite networks, including commercial constellations such as Starlink, OneWeb, and Amazon Kuiper, are becoming a mainstream option for broadband access where terrestrial infrastructure is unavailable or economically difficult to deploy [1]–[3]. Increasingly, the same reach also serves as a backhaul for remote Internet of Things (IoT) and edge deployments in regions that terrestrial networks do not cover [4]. Because they rely on fast-moving satellites, the serving satellite and gateway are reconfigured frequently, making a LEO access network far more dynamic than a terrestrial one. This dynamism appears not only as latency

and throughput variation but also as *packet reordering* and *packet loss* (we explain in detail in §IV). This is problematic for TCP, which was designed for terrestrial paths where loss is the primary congestion signal and is inferred from ACK feedback. On a LEO link, this inference misleads: reordering produces duplicate ACKs that resemble loss, and intermittent wireless loss occurs without queue buildup [5]. In both cases, TCP retransmits and shrinks its congestion window more than actually requires, leaving the available capacity underused.

In this paper, we measure and analyze a commercial Starlink and quantify how its reordering and loss affect widely deployed TCP congestion-control algorithms. Prior measurements have characterized Starlink's latency, throughput, and its roughly 15-second reconfiguration cycle [6]–[11]. However, two transport-layer questions remain open. First, packet reordering has rarely been measured directly, even though it readily makes TCP misinterpret delayed packets as lost ones. Second, loss has been characterized mainly by bit rate, leaving open whether it is also sensitive to packet rate [11].

We address these questions in three steps: we measure how reordering occurs, test whether loss is *rate-sensitive*, and compare three standard TCP loss-detection mechanisms—cumulative ACK (CAK) [12], selective ACK (SACK) [13], and the reordering-aware Recent ACKnowledgment (RACK) [14]—across eight state-of-the-art congestion-control algorithms. We combine controlled UDP experiments, which carry no congestion control and thus expose the channel's loss and reordering directly, with TCP experiments that link these disturbances to transport-layer responses across various congestion-control algorithms.

Through this measurement study, we make the following findings about how Starlink's characteristics disturb TCP:

- **Reordering is real and direction-dependent.** It occurs on both links, but through different mechanisms (§IV-A). Uplink reordering indicates a sub-millisecond scheduler/buffer effect: an out-of-order packet arrives only ≈ 0.3 ms late (median) and no loss accompanies it. Downlink reordering instead shifts a packet by ≈ 3 ms (median, the scale of one wireless-hop round trip) and is accompanied by loss, which link-layer recovery or queueing would produce.
- **Loss depends on packet rate, not the bit rate alone.** A single *per-packet service-time model* captures the measured capacity of both links. It shows numerically that capacity is set by both the packet rate and payload size, not by the bit rate alone (§IV-B). On the downlink, this yields a hard packet rate cap of $\approx 112,000$ packets per second (pps) for small payloads, well below the bit rate ceiling (≈ 296 Mbps).

All authors are with the Department of Computer Science and Engineering, Chung-Ang University, Seoul, Republic of Korea (email: alswns2631,hello0922,jpaek@cau.ac.kr). J. Paek is the corresponding author.

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2024-00359450), and also by the Institute of Information & Communications Technology Planning & Evaluation (IITP)-ITRC (Information Technology Research Center) grant funded by the Korea government (MSIT) (IITP-2026-RS-2022-00156353, 50%).

Copyright (c) 2026 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending a request to pubs-permissions@ieee.org.

The uplink is instead bound by an ≈ 18 Mbps token-bucket policer.

- **Reordering-aware detection helps.** RACK cuts the retransmission rate by ≈ 13 –52% (median ≈ 21 –42%), showing that reordering-induced spurious retransmissions are a substantial and recoverable inefficiency which motivates satellite-aware congestion control (§VI).

The remainder of this paper is organized as follows. We first review prior Starlink measurements to position our work (§II), and describe our testbed and experiment methodologies (§III). We analyze the channel itself with UDP (§IV), characterizing where and why reordering occurs and how loss depends on the packet rate. We then measure how these disturbances affect TCP across the eight congestion-control algorithms and three loss-detection mechanisms (§V). Finally, we discuss the implications for satellite-aware congestion control (§VI) before concluding the work (§VII).

II. RELATED WORK

We review prior work on Starlink and highlight the underexplored aspect addressed by this paper.

A first look at Starlink reported its latency, throughput, and loss as seen by end users [15]. Later work extended these measurements across regions and timescales [7], [9], [16]. Together they show that Starlink is usable, but far more variable than terrestrial broadband.

Internal dynamics. Starlink reconfigures its paths on a roughly 15-second cycle, and latency and throughput degrade at those boundaries [9]. Prior work inferred its scheduling architecture, including a global controller and on-satellite scheduling [17]. Further studies characterized bursty downlink allocation [18] and decomposed the one-way delay [6], [8].

Queueing and loss. Starlink access network applies drop-front rather than drop-tail buffering, and does not appear to enforce per-flow fair queueing [11]. This distinction matters for deciding whether observed loss reflects congestion or queue management. The same work further notes that loss depends on the packet rate as well as the byte rate. Load-induced RTT inflation and bufferbloat have also been reported [9].

Transport-layer impact. Comparisons of Linux congestion-control algorithms find that BBR variants generally outperform CUBIC on Starlink [19]. QUIC- and TCP-based studies report that reconfiguration-induced loss and latency spikes are often misread as congestion [10], [20], [21]. Emulation further shows that LEO route changes can cause out-of-order delivery and spurious TCP loss recovery [22].

Together, these studies establish how Starlink's latency, throughput, reconfiguration, and queueing behave. Yet they leave three transport-layer questions open. First, packet reordering caused by route and satellite changes has so far been studied only in emulation [22]; how often it actually occurs on a real commercial Starlink link, and through what mechanism, has not been measured directly. Second, this packet-rate dependence of loss has been noted but never quantified [11]. Third, how reordering and rate-sensitive loss together mislead TCP loss detection on real hardware remains unestablished. To address this missing perspective, we collect paired UDP and

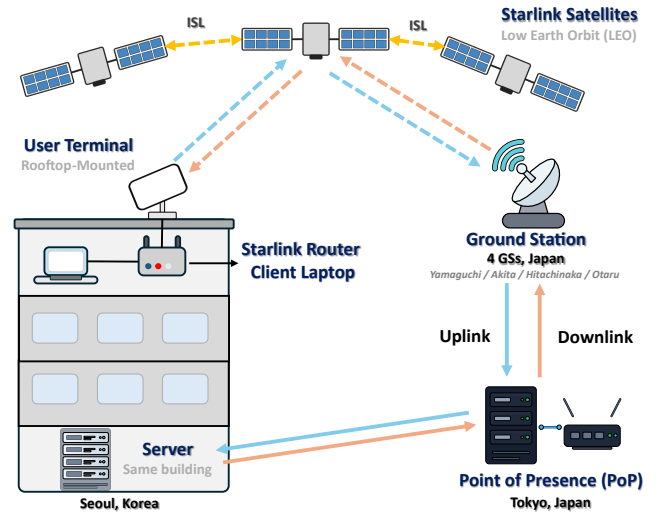


Fig. 1: Measurement testbed. A client behind a rooftop Starlink user terminal and a server on a wired campus network (both in Seoul, Republic of Korea) exchange traffic over the satellite path. Traffic from the client to the server traverses the *uplink* (user terminal $\rightarrow$ satellite $\rightarrow$ ground station (GS) $\rightarrow$ PoP), and traffic in the reverse direction the *downlink*. The serving PoP is in Tokyo, Japan, and the four GSs registered in Japan are the candidates for the serving one.

TABLE I: Testbed configuration.

Component	Specification
Server	Intel i7-7740X, 64 GB RAM, Linux 6.8.0
Client	Intel i5-10400F, 48 GB RAM, Linux 6.13.7
Dish	Starlink Standard (Gen 4), Residential plan

TCP packet captures on both the uplink and the downlink. We quantify reordering, the packet-rate sensitivity of loss, and their impact on each loss-detection mechanism.

III. MEASUREMENT METHODOLOGY

In this section, we describe our testbed and the controlled UDP and TCP experiments used to measure packet reordering, packet loss, and TCP performance on a real Starlink link.

Testbed. We measure a residential Starlink link with a Standard (Gen 4) dish mounted on a rooftop with an unobstructed sky view (Fig. 1) from March to August 2026. A client behind the dish and a server on a wired campus network exchange traffic in two directions: the *downlink* (server $\rightarrow$ client) and the *uplink* (client $\rightarrow$ server). On both directions, traffic crosses the satellite to a GS and an operator Point-of-Presence (PoP) before reaching the wired server, so every measured packet traverses the full satellite path rather than a local shortcut. Our terminal is in Seoul, Republic of Korea, and Starlink serves it from Japan: the path goes through the Tokyo PoP, about 1,153 km away, before returning to Korea. The four GSs Starlink has registered in Japan are the candidates for the serving one¹ (555–1,358 km from our site). Table I summarizes the hardware used for the measurements, and the

¹MIC (Japan) Radio Station Search, licensee “Starlink Japan G.K.”: <https://www.tele.soumu.go.jp/musen/SearchServlet> (accessed 2026-09-09).

TABLE II: UDP measurement parameters. Supplementary measurements used in specific analyses (e.g., denser downlink payload grid and longer uplink runs) are described in the relevant sections.

Parameter	Value
Payload size	100–1300 B (step 100 B), 1472 B
Uplink rate	5–45 Mbps (step 5 Mbps)
Downlink rate	50–350 Mbps (step 25 Mbps)
Duration	200 seconds
Repetitions	10 times

measurement dataset is publicly available on IEEE DataPort at <https://dx.doi.org/10.21227/bc71-sg44>.

UDP measurements. To expose the channel directly, we generate UDP traffic with a custom traffic generator we built, in which each packet carries a *sequence number* and a *send timestamp*. From these, the receiver derives three metrics: a missing sequence number is counted as *loss*, a packet is counted as *reordered* if its sequence number is lower than the next expected one [23], and the difference between the arrival time and the embedded send timestamp gives the *one-way delay*. Under this definition a packet that arrives beyond the expected sequence number is in order and merely opens a gap; only the packets that later arrive to fill it are reordered. Transient spikes in the one-way delay also let us identify the *path reconfiguration* events that recur on Starlink’s roughly 15-second cycle. Because the sender and receiver clocks are independent, we correct the one-way delay using a separate wired channel: the two hosts are also joined by a direct campus Ethernet path, and we run NTP over it to estimate the clock offset, which we subtract from each delay sample. We synchronize over this wired path rather than over Starlink whose high variability makes it unsuitable for clock synchronization.

Each run lasts 200 seconds, long enough to span at least ten reconfiguration events. Across both directions, we sweep the full set of 14 payload sizes, spanning 100 B up to the 1472 B maximum, together with the per-direction sending rates listed in Table II. We repeat every rate–payload combination 10 times in randomized order so that no single setting is biased toward a particular time or network state. This full payload range is what lets us separate the packet-rate-bound small payloads from the bit-rate-bound large ones. On top of this full sweep, we additionally lay a finer grid over the small-payload region on the downlink, sampling from 100 to 400 B in 10 B steps; these extra points only refine the location of the packet-rate cap and do not replace the full-range measurement.

TCP measurements. TCP experiments run on the uplink only (Starlink client → campus server); we defer the rationale for excluding the downlink to §V-A. Each experiment is a single `iperf3` 3.9 flow of 300 seconds, repeated 10 times. We compare eight congestion-control algorithms that span the major families: loss-based Reno [24] and CUBIC [25], delay- and bandwidth-based Vegas [26] and Westwood [27], model-based BBRv1 [28] and BBRv3 [29]², and the LEO-specific

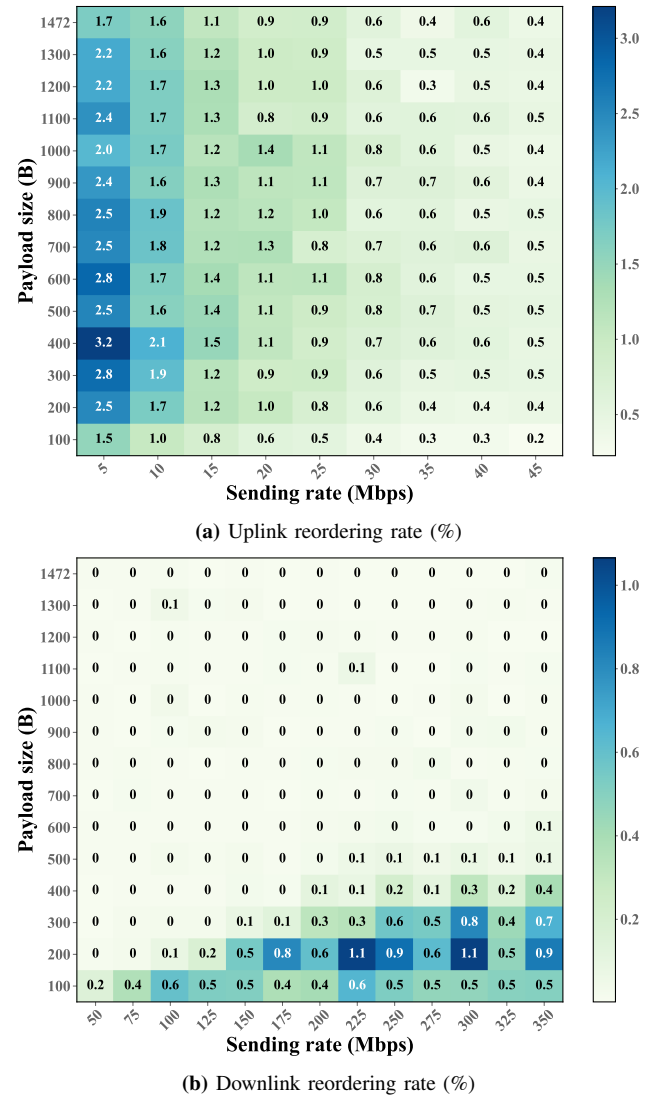


Fig. 2: Reordering rate (%) across sending rate and payload size. Uplink reordering concentrates at low sending rates and is largely independent of payload, whereas downlink reordering is rare overall but rises sharply as the payload shrinks and the packet rate grows.

LeoCC [21]³ and SatPipe [10]⁴. Each algorithm is loaded as a Linux kernel module, so the default and custom algorithms run under identical conditions.

Orthogonally, we test three loss-detection modes to isolate how each mechanism reacts to the channel’s reordering and loss: cumulative-ACK recovery with selective ACK disabled (`tcp_sack=0`, denoted CACK), which follows TCP’s cumulative acknowledgment semantics [30] and duplicate-ACK based fast retransmit/recovery [12]; classic SACK-based recovery [13] (`tcp_sack=1` with RACK off); and time-based RACK recovery [14]. From sender-side packet captures, we compute goodput, throughput, and the retransmission rate in 1-second windows, and from kernel logs, we record the sample RTT, congestion window, and in-flight bytes on every ACK. We verified that TCP could open a window larger than the

²<https://github.com/google/bbr>

³<https://github.com/SpaceNetLab/LeoCC>

⁴<https://github.com/dzhao99/SatPipe>

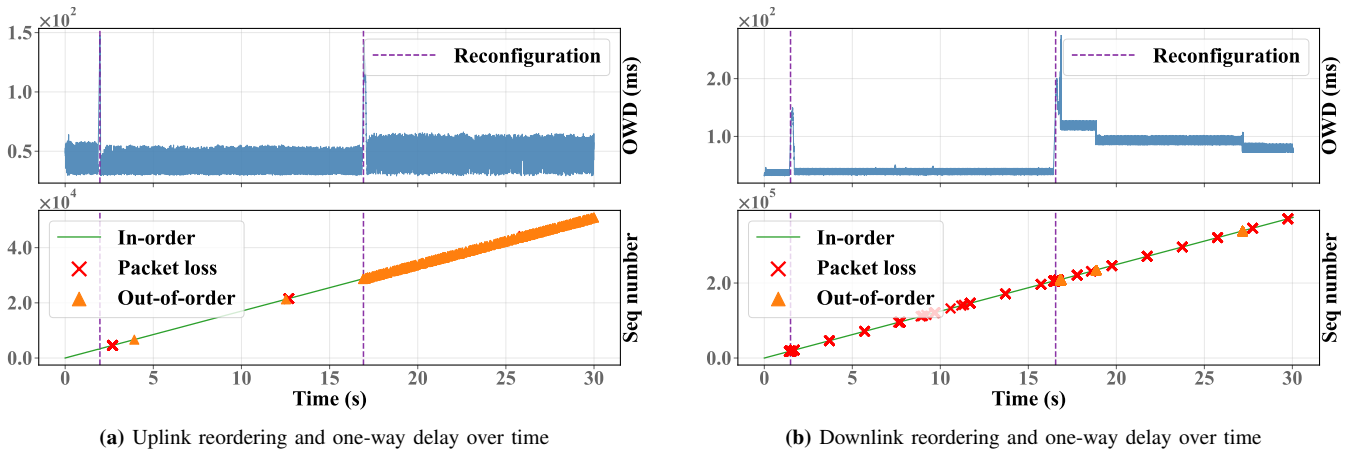


Fig. 3: Reordering and one-way delay (OWD) over time. On both links, the out-of-order pattern shifts at the path reconfiguration boundaries; on the uplink the events also arrive in bursts rather than uniformly, which argues against a static terrestrial cause.

path's bandwidth–delay product, so the throughput differences we report reflect congestion-control and loss-recovery behavior rather than socket-buffer limits.

IV. UDP ANALYSIS

We use controlled UDP measurements to characterize the two channel disturbances that later mislead TCP: packet reordering and loss. Because UDP has no congestion control, it exposes the channel as is, with every packet delivered, dropped, or reordered by the network alone without transport-layer reaction to confound the measurement. We first establish where reordering occurs and what mechanism produces it (§IV-A), and then analyze packet loss (§IV-B).

A. Packet Reordering

Candidate mechanisms. We consider five hypotheses about how packets could be reordered: reordering on the terrestrial part of the path (H1) [31]; a path reconfiguration, after which packets on the old path arrive late (H2) [22]; link-layer recovery on one of the wireless hops (H3); segmentation of a packet across radio frames (H4) [32]; and parallel queues inside the satellite segment (H5). For H5, a Starlink patent describes the MAC layer as keeping one queue per modulation and coding scheme (MCS), so packets in different queues can leave out of order [33]. Each hypothesis would leave a different signature, so we check each against the measurements below and set aside those they contradict.

Where reordering occurs. We begin by mapping how often packets arrive out of order. Fig. 2 plots the reordering rate (%) as a function of sending rate and payload size. The two traffic directions behave in opposite ways. On the uplink (Fig. 2(a)), reordering is governed by the sending rate: it is highest—a few percent of all packets—when the rate is low, and falls steadily as the rate increases. On the downlink (Fig. 2(b)), reordering is rare in absolute terms but follows the opposite trend, growing as the payload size shrinks. The fact that the two directions respond to different variables is the first hint that their reordering arises from distinct causes.

Reordering follows the satellite path. We next ask whether reordering originates on the satellite link or somewhere in the fixed terrestrial segment between the gateway and our server. Fig. 3 plots out-of-order events over time. On both links, the reordering pattern changes abruptly at the path reconfiguration boundaries, which recur on Starlink's roughly 15-second cycle, and on the uplink (Fig. 3(a)) the events further arrive in bursts rather than being spread uniformly; reordering also appears between reconfigurations (weakening H2). Were the terrestrial segment responsible, reordering would appear uniformly and would be insensitive to satellite handovers, since the terrestrial part of the path would change far less than the satellite segment. Because the rate and timing of reordering track the reconfiguration cycle and cluster into bursts, we attribute it to the satellite (weakening H1).

Gap-fill time. Assuming that reordering happens on the satellite link, we now ask through what mechanism, and we use the timing of each out-of-order event to tell the candidates apart. When a packet arrives whose sequence number is higher than expected, it opens a *gap*. We define the *gap-fill time* as the interval from the moment the gap opens until the missing lower-numbered packet finally arrives and closes it. The distribution of this interval discriminates between the mechanisms that could produce reordering on a satellite link. Suppose reordering is caused by signal loss on one of the wireless hops that is then repaired by link-layer recovery (H3). The late packet must then traverse that hop a second time, so the gap-fill time should be on the order of one satellite–user-terminal round trip (roughly 3–10 ms), and reordering should frequently coincide with loss. If instead reordering came from frame segmentation (H4), a split packet would be completed only when the next frame arrives, so the gap-fill time should be at least the 1.33 ms frame period [18], and no loss would be involved.

Which mechanism dominates. To apply the test, we measure two quantities per direction: the fraction of gaps that are ever closed—which we call *gap resolution ratio*—and, for those that close, the *gap-fill time* distribution. *Gap resolution* reflects pure reordering, whereas a gap that never closes means the

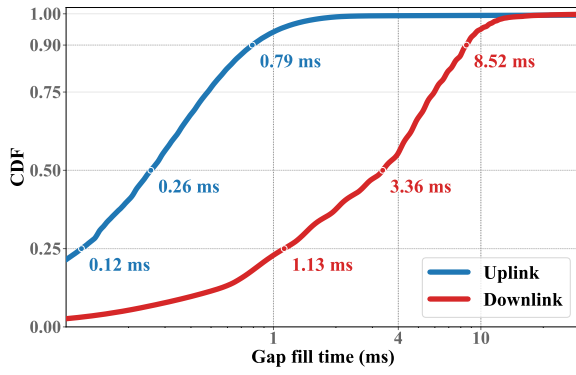


Fig. 4: Gap-fill time CDF. Uplink gaps close in sub-millisecond times, whereas downlink gaps close on the scale of a satellite–user–terminal round trip.

missing packet was genuinely lost, so reordering and loss occurred together. The two directions land on opposite ends of both measures. The uplink resolves 94% of its gaps, and its gap-fill times are sub-millisecond, with a median of 0.26 ms and a 90th percentile of 0.79 ms (Fig. 4). This matches the intra-satellite hypothesis (H5): gaps open and close almost immediately, and almost none are accompanied by loss. These numbers weaken H3 and H4 on the uplink, where the sub-millisecond gap fills sit far below both the hop round trip and the 1.33 ms frame period. The downlink, in contrast, resolves only 34% of its gaps. The gaps that do close take far longer—a median of 3.36 ms and a 90th percentile of 8.52 ms—consistent with the satellite–user–terminal round-trip scale predicted for link-layer recovery (H3). We also set H4 aside on the downlink, since segmentation gives no reason for the loss that accompanies reordering.

Summary. The two links therefore appear to reorder packets for different reasons. Uplink reordering is a loss-free, sub-millisecond effect, which is consistent with internal processing within the satellite segment (H5). The out-of-order packets therefore always arrive; they merely arrive slightly early relative to one another. If a reconfiguration also changes the state of the ground side, H1 remains a possibility there as well, and some events do occur right at a reconfiguration, which keeps H2 partly supported. Downlink reordering can be explained as a by-product of signal loss on one of the wireless hops that link-layer recovery (H3) repairs only in part: most gaps are never filled, and the ones that are filled take a full round trip. Queueing inside the satellite segment (H5) can leave the same trace, since the downlink queue is the smaller one [34]: a queue buffer overflow drops the packets left behind. We summarize the verdict on each candidate in Table III.

B. Packet loss

Having separated reordering from loss, we now ask where and why packets are dropped. We find that loss is not governed by the bit rate alone but by the rate at which the link must process individual packets (packets per second, pps), and that a single per-packet service-time model reproduces this behavior on both directions.

TABLE III: Candidate mechanisms for the reordering we observe. The observation that decides each row is given in the text.

#	Candidate mechanism	Up	Down
H1	Terrestrial ECMP	●	△
H2	Path reconfiguration	●	●
H3	Link-layer recovery	△	●
H4	Frame segmentation	△	△
H5	Parallel queues in the satellite	●	●

● supported ● partly supported △ weakened

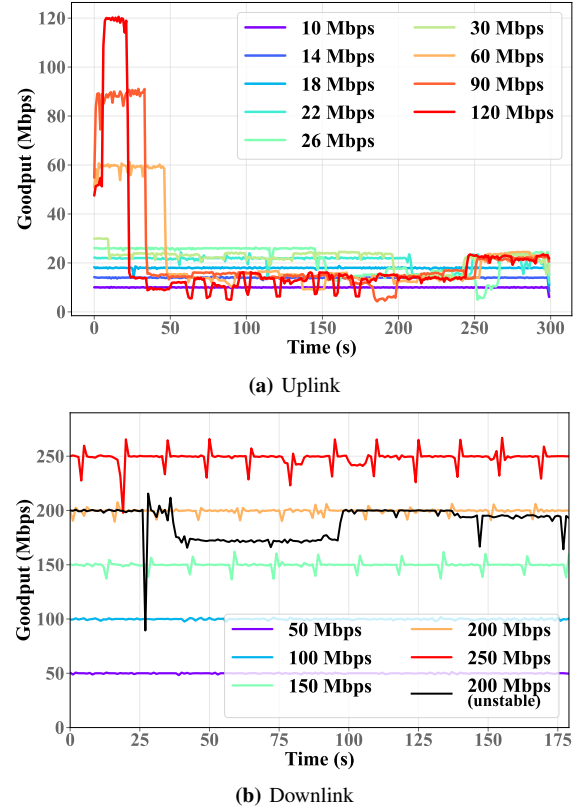


Fig. 5: Goodput over time. When the offered load is raised, the uplink passes an initial burst on accumulated tokens and then collapses to a sustained rate of about 18 Mbps, the signature of a token-bucket policer. The downlink behaves differently: it is driven at several application rates and the goodput tracks each of them. The black trace is one 200 Mbps run in which the link itself degrades.

Traffic shape and policing. Before mapping loss, we first look at how each direction shapes the offered traffic, since this shaping determines when loss starts. The two directions shape traffic in different ways (Fig. 5). The uplink behaves like a *token-bucket policer*. When we raise the offered load to 90 Mbps or 120 Mbps, the link first passes a burst for about 20 s using tokens accumulated in the bucket (Fig. 5(a)). Once the tokens drain, the goodput drops to about 18 Mbps and stays there, the behavior expected of a token bucket. The downlink shows no such behavior: its goodput mostly follows the application rate, though at high rates the goodput fluctuates when the link degrades (Fig. 5(b)). We also note that the downlink truncates these flows early: although each run is offered for 200 s, delivery consistently stops once about 180 s

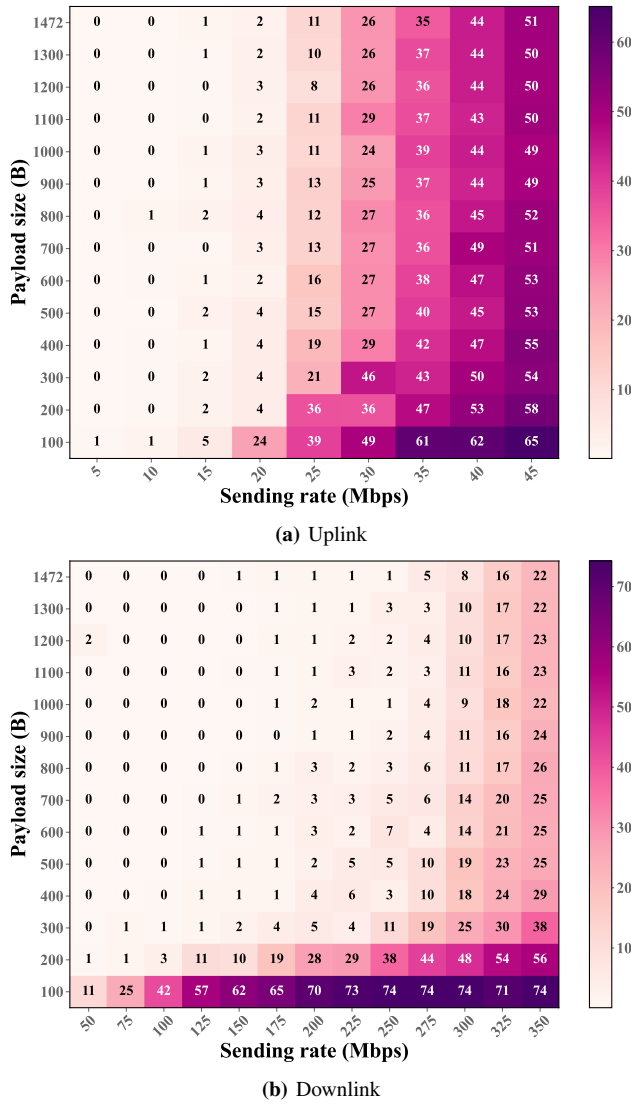


Fig. 6: Loss rate (%) across sending rate and payload size. Loss is not a function of the bit rate alone: on both directions small payloads lose much earlier than large ones, and on the uplink the largest payloads are ultimately capped near the token rate $R \approx 18$ Mbps.

have passed since that flow's own first packet, independent of payload size and offered rate. This is not a one-off artifact: across all downlink runs, delivery never extended past about 180s, so the cutoff is the rule rather than the exception. Starlink delivers its residential service behind a carrier-grade NAT [35], and this cut is likely to be the expiry of the NAT mapping: the cut applies to each flow separately, and sending even a small packet in the reverse direction keeps the flow alive, which matches how a NAT mapping is refreshed by outbound traffic [36].

We estimate the token-bucket parameters from the uplink runs, which run long enough (200s) for the accumulated tokens to drain. With the bucket full, we release a burst and read the sustained rate R from the end-slope of the burst run. From a byte-saturated run, we then convert the area held during the burst into the bucket capacity B as,

$$B = (r_{\text{peak}} - R)t^*,$$

where r_{peak} is the peak rate sustained during the burst, and t^* is how long that burst lasts before the goodput collapses to R . This yields $R \approx 18 \pm 2$ Mbps and $B \approx 278 \pm 41$ MB. That R is independent of both sending rate and payload supports reading it as a fixed token-replenishment rate.

On the uplink, B varies considerably from run to run, so, to reduce the bucket's influence and obtain a stable estimate, we first drain the bucket for about 20s and then measure over the following 30s.

Loss grows with packet rate, not just bit rate. We next map the loss rate over the operating range of each link. Fig. 6 plots loss against sending rate and payload size, and a single bit-rate axis fails to explain it.

On the downlink (Fig. 6(b)), large payloads (≥ 1000 B) stay near-lossless until the sending rate passes about 300Mbps, so they are bit-rate-bound. Small payloads instead lose heavily from low sending rate upward: a 100 B payload already drops 11% at 50Mbps, 42% at 100Mbps, and 70% at 200Mbps, so they are pps-bound. These two patterns line up not along bit rate but along packet rate: at a fixed bit rate, a smaller payload sends more packets per second, and it is exactly these high-pps flows that lose the most, while large-payload flows at the same bit rate do not. Loss therefore tracks the packet rate, not the bit rate alone.

The uplink shows the same payload ordering once the bucket has drained, so we plot only its steady-state region (Fig. 6(a)). A 100 B payload already drops 11% at 15Mbps and 39% at 25Mbps, whereas a 1472 B payload stays near-lossless up to about 20Mbps before its loss climbs. Large uplink payloads are ultimately capped near the token rate $R \approx 18$ Mbps, but the fact that small payloads lose far earlier shows that the uplink, too, is governed by the packet rate rather than by a single bit-rate ceiling. This leaves a quantitative question—at what packet rate does each link begin to drop?—which motivates the model below.

A single service-time model. To fold the bit-rate and packet-rate effects into one description, we model the time to serve one packet as a fixed cost plus a cost proportional to its size,

$$T(L) = \alpha + \beta L \quad [\text{s/pkt}], \quad (1)$$

where L is the payload in bits, α is the fixed per-packet cost (the same for any packet, such as header and scheduling overhead), and β is the per-bit cost. The same two costs give the packet-rate and bit-rate capacities directly,

$$C_{\text{pps}}(L) = \frac{1}{\alpha + \beta L} [\text{pkt/s}], \quad C_{\text{bps}}(L) = \frac{L}{\alpha + \beta L} [\text{bps}]. \quad (2)$$

The two extremes are easy to read off. A very small packet ($L \rightarrow 0$) costs almost only α , so the link tops out at $1/\alpha$ packets per second—its *maximum packet rate*. A very large packet ($L \rightarrow \infty$) is dominated by the per-bit term, so its throughput tops out at $1/\beta$ bits per second—its *maximum bit rate*. Small payloads are therefore limited by the packet rate and large payloads by the bit rate, so this single linear cost reproduces both regimes of the heatmap.

Validating the model. Each payload is measured at many sending rates, so its samples include runs that sit well below

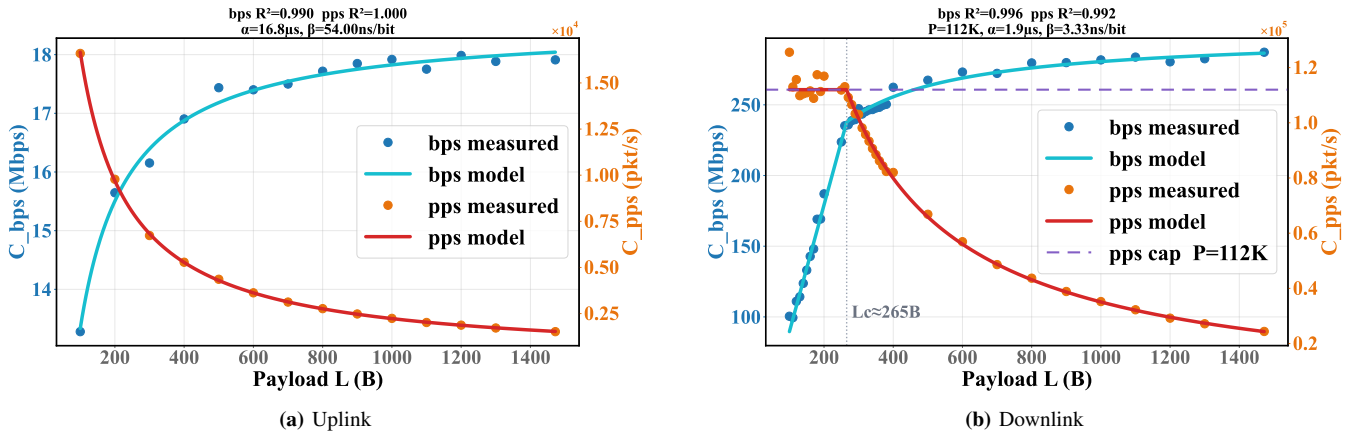


Fig. 7: Measured capacity versus the service-time model $T(L) = \alpha + \beta L$. The model reproduces both the packet-rate and bit-rate capacities ($R^2 \geq 0.99$). On the downlink, a hard packet-rate cap $P = 112$ Kpps flattens C_{pps} for payloads below $L_c \approx 265$ B; on the uplink the capacity rises with payload toward a maximum bit rate of $1/\beta \approx 18.6$ Mbps, close to the token rate $R \approx 18$ Mbps.

capacity. To estimate the capacity itself, we keep for each payload the top decile of samples by achieved packet rate, treat these near-saturation points as the measured capacity, and compare them against the two formulas (Fig. 7); if the model reproduces this near-saturation envelope, we take Starlink to follow the service-time model. As the evaluation metric for this fit, we use the coefficient of determination;

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}, \quad (3)$$

where y_i is the measured capacity, $\hat{y}_i$ the model prediction, and $\bar{y}$ the mean of the measurements; R^2 approaches 1 as the model explains more of the variance, and $R^2=1$ is a perfect fit [37].

On the downlink (Fig. 7(b)), the model is accurate (bps $R^2=0.996$, pps $R^2=0.992$) with $\alpha=1.9\mu s$ and $\beta=3.38 ns/bit$. Below about 300 B, however, the downlink departs from the service-time curve, so we sample this region densely (payloads from 100 to 400 B in 10 B steps, the additional points in the figure) to locate the cap accurately. A grid search over candidate caps then localizes a hard packet-rate ceiling at $P=112$ Kpps, and for payloads below $L_c \approx 265$ B, the C_{pps} curve flattens against this cap rather than against the service-time limit.

The uplink (Fig. 7(a)) follows the same model almost exactly, with both fits strong (pps $R^2=1.000$, bps $R^2=0.993$) at $\alpha=16.4\mu s$ and $\beta=53.8 ns/bit$. Here β is about $16\times$ the downlink value, so each bit is far more expensive to send: the capacity climbs smoothly with payload toward a maximum bit rate of $1/\beta \approx 18.6$ Mbps, which nearly coincides with the token rate $R \approx 18$ Mbps. The two limits therefore agree for large payloads, while small payloads fall well below R because the per-packet cost α binds first; unlike the downlink, the uplink needs no separate packet-rate cap, as the service-time model alone already captures its capacity across all payloads.

What binds each direction. The fitted parameters pin down which resource saturates first (Table IV). On the downlink, the packet-rate cap (112 Kpps) bites small packets before the 296 Mbps bit-rate ceiling does, and $L_c \approx 265$ B marks the

TABLE IV: Binding resource per direction, from the fitted model parameters.

Dir.	Max pps ($1/\alpha$)	Max bps ($1/\beta$)	Extra limit / binding
Down	526 Kpkt/s	296 Mbps	pps cap $P = 112K$ (small L : pps, large L : bit rate)
Up	61 Kpkt/s	18.6 Mbps	token rate $R \approx 18$ Mbps (small L : pps, large L : token)

boundary between the pps-bound and bit-rate-bound regimes. On the uplink, the maximum bit rate $1/\beta \approx 18.6$ Mbps and the token rate $R \approx 18$ Mbps nearly coincide, so large payloads meet both at once while small payloads are limited earlier by the per-packet cost; the capacity rises smoothly with payload, with no separate cap or knee.

Summary. A single service-time model $T(L) = \alpha + \beta L$ explains loss on both directions at $R^2 \geq 0.99$. The fitted α , β , packet-rate cap P , token rate R , and bucket size B together pin down, per direction, which resource saturates and where loss begins. Loss is therefore structured rather than random: on both directions, small, high-packet-rate traffic predictably meets these limits, as does any uplink traffic above about 18 Mbps.

Takeaways. Across both analyses, the two disturbances that later mislead TCP are deterministic and direction-dependent. On the downlink, reordering appears together with loss; loss is governed by a packet-rate cap and strikes small, high-rate packets hardest. On the uplink, packets are reordered, likely by a buffering effect, and are instead lost to a token-bucket policer. These mechanisms are also stable over time: when we partition each measurement into 3-hour windows and recompute the loss heatmaps and gap-fill times per window, the results vary little across windows, so the effects are not artifacts of a particular time of day; nor of varying network load, since the weekly mean capacity moved by about 3% over the campaign (236–242 Mbps on the downlink, 17.7–18.3 Mbps on the uplink). Because some of these effects resemble congestion signals, a TCP sender may back off more than necessary; the next section measures how much.

V. TCP ANALYSIS

The UDP measurements (§IV) showed that the Starlink channel reorders and drops packets. We now ask how these disturbances affect real TCP traffic—specifically, how each pairing of a congestion-control algorithm with a loss-detection mechanism reacts to them. We study the uplink only: §V-A explains why the downlink is unsuitable for an end-to-end congestion-control study, and §V-B and §V-C then analyze, respectively, how the congestion-control algorithm and the loss-detection mechanism respond to reordering and loss on the uplink.

A. Why we exclude the downlink

On the downlink, the path does not behave like a single end-to-end TCP connection. Satellite operators commonly insert a performance-enhancing proxy (PEP) that splits the connection and runs congestion control on the provider's side [38], and three observations suggest that the downlink is not carried end to end here either.

Evidence 1: the handshake is rewritten. We capture the same downlink connection simultaneously at the sender (our server) and at the receiver (the user terminal). The TCP sequence and acknowledgment numbers and the TCP Timestamps-option values (TSval/TSsec) of the handshake differ between the two capture points, whereas on the uplink they are identical. Each side is internally self-consistent—its SYN-ACK acknowledges its own SYN—but the two sides use unrelated sequence numbers, and even the timestamps are shifted by a fixed offset. The handshake our server sees is therefore not the one the user terminal sent.

Evidence 2: the RTT is implausibly short. The kernel reports a downlink RTT of only 1–2 ms, far below the 80–100 ms measured on the uplink. An RTT this short is far too low for a satellite hop, which suggests that our server's segments are acknowledged well before they reach it.

Evidence 3: the receive window is rewritten. The receive-window (*rwnd*) series captured at the server and at the user terminal diverge completely (Fig. 8), so the window our server sees is not the one the user terminal advertises.

Taken together, these three signs—a rewritten handshake, a 1–2 ms RTT, and server- and client-side *rwnd* series that disagree—show that the downlink connection is not carried end to end. Several devices could produce similar signs (Table V). Because the algorithm we configure therefore does not control the downlink end to end, we exclude the downlink and focus the rest of our TCP analysis on the uplink, where the end host remains in control.

B. Impact on Congestion Control

On the uplink, the end host's algorithm is in control, so we compare all eight congestion-control algorithms under the three loss-detection modes. Fig. 9 reports the goodput–RTT trade-off and Fig. 10 the retransmission rate of every combination. The algorithms fall into three groups that respond very differently.

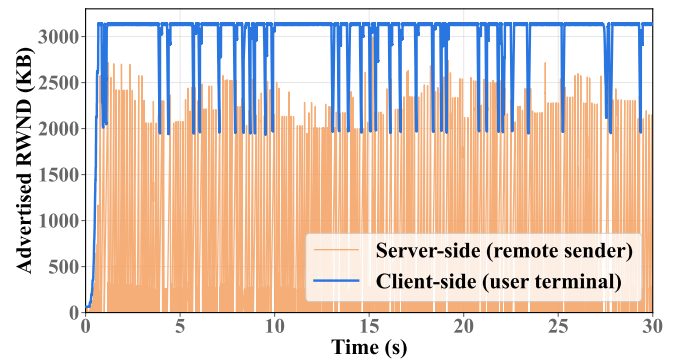


Fig. 8: Receive window (*rwnd*) over time, captured at the server and at the user terminal for the same downlink flow. The two series diverge, so the window the server sees is not the one the terminal advertises.

TABLE V: Middlebox behaviors that could produce signs similar to those we observe on the downlink. Only the last one produces all three. ALG: application-level gateway, which rewrites addresses inside the payload.

Device	Sign it can produce
SYN proxy [39]	Handshake answered nearby
TCP normalizer [40]	Sequence numbers, timestamps rewritten
NAT with ALG [41]	Sequence numbers rewritten
Traffic shaper [42]	Receive window clamped
ACK filtering [43]	RTT estimate distorted
Split-TCP proxy [44]	All of the above

Delay-based. Vegas performs worst of all (Fig. 9). As our one-way-delay measurements showed (Fig. 3(a)), the Starlink RTT fluctuates constantly, driven by reordering, queueing, and the 15-second reconfiguration cycle. Reading this ordinary variation as congestion, Vegas keeps backing off, so it severely underutilizes the link and lands at by far the lowest goodput of any algorithm.

Loss-based. Reno, CUBIC, and Westwood are almost insensitive to the loss-detection mode. Their retransmission rate stays near zero under all three modes (Fig. 10) because the uplink rarely loses packets and they leave the link underutilized in every case. With so few loss events to detect, the loss-detection mechanism has little to act on, so switching among CACK, SACK, and RACK barely changes their behavior.

Model-based. BBRv1, BBRv3, LeoCC, and SatPipe, in contrast, are strongly affected by the loss-detection mode. These algorithms drive sending from sampled RTT and bandwidth, and the validity of those samples depends on how the incoming ACK stream is interpreted. The way reordering and loss are detected therefore shapes their behavior directly.

C. Impact of Loss detection

Holding the congestion-control algorithm aside, we now investigate the impact of the three loss-detection modes on uplink behavior. Because the model-based algorithms are the ones that react to the mode, we focus on them and compare the modes in two steps: CACK against SACK, then SACK against RACK.

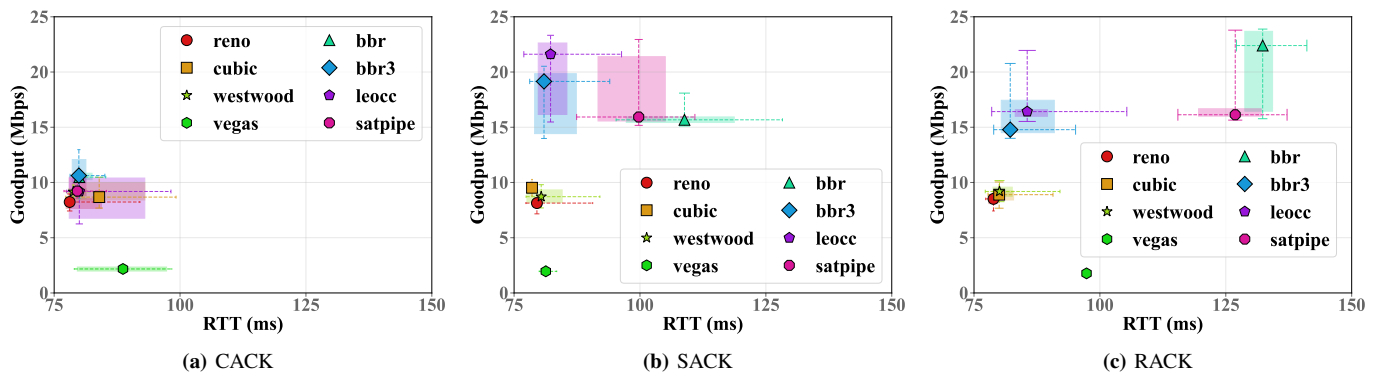


Fig. 9: Goodput vs. RTT for all eight congestion-control algorithms under each loss-detection mode (higher and farther left is better). The model-based algorithms (BBR, BBRv3, LeoCC, SatPipe) shift markedly across modes, whereas the loss-based ones and Vegas barely move.

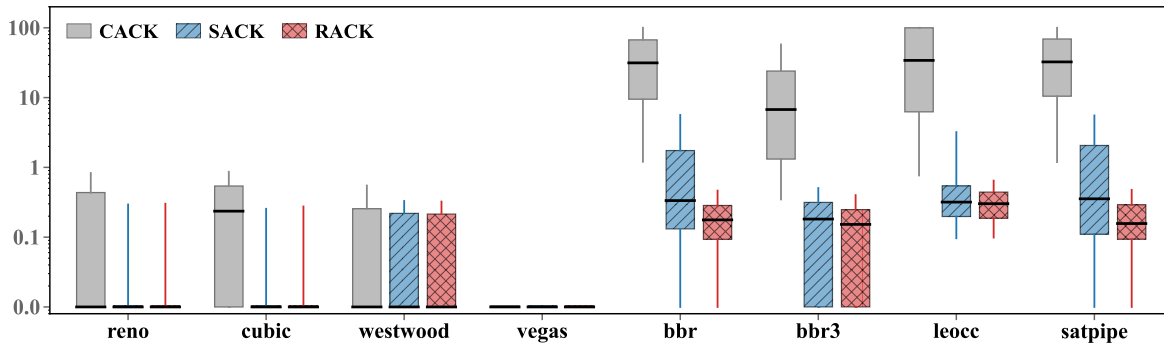


Fig. 10: Retransmission rate (% , log scale) by algorithm and loss-detection mode. The model-based algorithms retransmit heavily under CACK; SACK and especially RACK cut this sharply, whereas the loss-based algorithms and Vegas stay near zero throughout.

CACK vs. SACK. Neither cumulative ACK nor selective ACK can distinguish reordering from loss, but CACK is far worse for the model-based algorithms. Two sampling rules explain the gap. Karn's algorithm forbids taking an RTT sample from a segment covered by duplicate ACKs [45], and the BBR rate sampler likewise discards bandwidth samples drawn from duplicate ACKs [29]. With selective ACK disabled, the channel's frequent reordering turns most acknowledgments into duplicates: under CACK, we measure a duplicate-ACK ratio around 80%. Without valid RTT and bandwidth samples, the model-based senders cannot gauge the link and overshoot it. Under CACK, they retransmit around 30% of their data, and their goodput collapses (Fig. 10). Enabling SACK restores most of the samples, which brings the retransmission rate back below 1% and recovers their goodput.

SACK vs. RACK. RACK uses per-segment timing to tell reordering from genuine loss, so it avoids the needless retransmissions that SACK triggers when packets merely arrive out of order. Relative to SACK, enabling RACK reduces the retransmission rate of the model-based algorithms by ≈ 13 –52% in the mean and ≈ 21 –42% in the median (Fig. 10). This robustness is not free. To avoid mistaking reordering for loss, RACK holds back a loss declaration until its reordering window expires, so a *genuinely* lost packet waits up to about one RTT before it is retransmitted [14]. The overhead grows with the RTT, since the window scales with it, yet the uplink reordering it guards against is only sub-millisecond (§IV-A) to begin with.

As a result, RACK improves the goodput–RTT trade-off for none of the four model-based algorithms (Fig. 9): two simply lose goodput, one keeps the same goodput only by running at a higher RTT, and the one that does gain goodput pays for it with a higher RTT as well. Which case a sender falls into depends on three of its properties, which Table VI lists. Its *rate cap* is whether it bounds its own rate estimate (for example BBRv3's *inflight_hi* limit); its *RTT guard* is whether it slows down as the measured RTT rises; and its *handover handling* is whether it takes any explicit action at a path reconfiguration.

BBRv1 has neither a cap nor an RTT guard, so the extra data RACK keeps in flight simply piles up as a queue. Its goodput rises, but the standing queue inflates the RTT, so this is a queue-loading artifact rather than a real efficiency gain. BBRv3's *inflight_hi* cap holds its rate down, so it cannot push harder to offset RACK's added recovery delay, and its goodput falls. LeoCC suffers the most: the extra in-flight data lengthens the RTT, its strong RTT guard reads this as congestion, and it throttles itself, so its goodput drops the furthest. SatPipe flushes its queue every 15 seconds to handle reconfigurations, which prevents any BBRv1-style queue gain, so its goodput stays flat while its RTT still creeps up. In short, on this loss-free uplink with only sub-millisecond reordering, RACK buys a lower retransmission rate but no better goodput, so throughput must always be read together with RTT.

TABLE VI: Sender properties of the four model-based algorithms and how their goodput responds, relative to SACK, when RACK is enabled. We explain each outcome in the text.

CCA	Rate cap	RTT guard	Handover	Goodput (RACK)
<i>BBRv1</i>	None	None	None	Rises (higher RTT)
<i>BBRv3</i>	Yes	Weak	None	Drops
<i>LeoCC</i>	Yes	Strong	Yes	Drops
<i>SatPipe</i>	None	None	Yes	Flat (higher RTT)

D. Takeaways

On the downlink, the connection is not carried end to end, so the end host's choice of algorithm has little effect there. On the uplink, where the end host is in control, the loss-detection mechanism matters mainly for the model-based algorithms. SACK is essential to keep their rate samples valid, and RACK then removes the spurious retransmissions that reordering causes. But uplink reordering is sub-millisecond and almost loss-free, so RACK's benefit shows up as a lower retransmission rate rather than higher goodput. As a result, no single algorithm both fills the link and keeps the RTT low. This gap motivates a satellite-aware design that pairs RACK's reordering robustness with an explicit rate cap and RTT guard, which we discuss in §VI.

VI. DISCUSSION AND LIMITATIONS

In this section, we discuss what our measurements imply for new congestion-control algorithms over Starlink and where our study is limited.

Toward satellite-aware congestion control. Because reordering and loss seem to be characteristics of the link (§IV), the loss-detection mechanism—not only the congestion-control algorithm—decides how well a sender copes, and no single default fits. Capacity is moreover bounded by the packet rate, not the bit rate alone, so a sudden rise in packet rate can cause loss even below the bit-rate ceiling. Our measurements point to three properties such a design should have. First, it must track the link's frequent reconfigurations: like SatPipe [10], LeoCC [21], and RedHat [46], it should continually re-estimate the path's capacity and delay, which shift at every reconfiguration rather than staying fixed. Second, it needs a loss detector matched to a long-RTT, reordering-prone channel. SACK is indispensable, but RACK's RTT-scaled timer is too costly once the RTT is large. So a duplicate-ACK threshold widened by a small safety margin—enough to absorb the channel's sub-millisecond reordering and layer-2 recovery—is a better fit. Third, it should budget the packet rate alongside the bit rate, and in particular avoid abrupt congestion-window jumps that spike the packet rate; tracking the window in both bytes and packets per link would let a sender respect the packet-rate cap directly.

Implications for IoT backhaul. Studies of low-power IoT networks usually treat the wide-area backhaul as a reliable pipe and concentrate their optimization on the constrained access side [47], [48]. Our results complicate that assumption when the backhaul is a LEO link. Capacity is bounded by packet rate

and small packets are penalized first. Two independent properties of such deployments combine to produce exactly this adversarial profile. First, IEEE 802.15.4 frames are capped at 127 bytes—a 102-byte MAC payload after framing overhead, and as little as a few dozen bytes once 6LoWPAN and RPL headers are added—so the packets are small [49]. Second, a 6LoWPAN/RPL gateway aggregates many leaf nodes, each periodically reporting sensor readings and routing-control traffic, and therefore forwards a high volume of these small frames onto the backhaul link [47], [48]. This is precisely the high-packet-rate, small-payload mix that loses the most and reaches the lowest goodput over Starlink (§IV-B). Provisioning such a deployment by bit rate alone would overestimate its usable capacity, so the packet rate must be budgeted as well.

Limitations. We measure only at the two endpoints, which lets us see that the downlink is not carried end to end but not which device intervenes (Table V) or how packets are reordered inside the satellite segment (§IV-A); measuring from additional vantage points is left as future work. We therefore cannot evaluate end-host TCP on the downlink; whether a UDP-based transport such as QUIC can run end to end is left open. Our results also come from a single terminal and location, and validation across more terminals, locations, and times remains future work. The absolute capacity may also shift as the constellation and the service evolve, so the numbers we report characterize the period we measured.

VII. CONCLUSION

We measured a commercial Starlink link and found that its reordering and loss are direction-dependent. Its loss is governed by the packet rate rather than the bit rate alone, a behavior captured by a single per-packet service-time model. On the transport layer, the loss-detection mechanism—not only the congestion-control algorithm—determines how well a sender copes: RACK removes the spurious retransmissions that CACK and SACK incur. These results indicate that transport over LEO networks is shaped by reordering and packet-rate effects as well as capacity and delay, and they motivate a satellite-aware congestion control that pairs reordering-robust loss detection with packet-rate-aware pacing.

REFERENCES

- [1] Starlink, "Starlink is connecting more than 10M active customers with high-speed Internet across 160 countries, territories and many other markets," accessed: 2026-06-15. [Online]. Available: <https://x.com/Starlink/status/2022446814591615013>
- [2] Eutelsat, "OneWeb LEO Constellation," 2026, accessed: 2026-06-15. [Online]. Available: <https://www.eutelsat.com/satellite-network/oneweb-leo-constellation>
- [3] Amazon, "Amazon Leo," 2026, accessed: 2026-06-15. [Online]. Available: <https://www.aboutamazon.com/what-we-do/devices-services/amazon-leo>
- [4] X. Gao, R. Liu, and A. Kaushik, "Virtual network function placement in satellite edge computing with a potential game approach," *IEEE Transactions on Network and Service Management*, vol. 19, no. 2, pp. 1243–1259, 2022.
- [5] S. Bhandarkar, A. L. N. Reddy, M. Allman, and E. Blanton, "Improving the Robustness of TCP to Non-Congestion Events," RFC 4653, Aug. 2006.

- [6] J. Pan, J. Zhao, and L. Cai, "Measuring a Low-Earth-Orbit Satellite Network," in *Proceedings of the IEEE 34th Annual International Symposium on Personal, Indoor and Mobile Radio Communications*, 2023, pp. 1–6.
- [7] J. Garcia, S. Sundberg, G. Caso, and A. Brunstrom, "Multi-Timescale Evaluation of Starlink Throughput," in *Proceedings of the 1st ACM Workshop on Low Earth Orbit Networking and Communication*, 2023, pp. 31–36.
- [8] J. Garcia, S. Sundberg, and A. Brunstrom, "A Detailed Characterization of Starlink One-way Delay," in *Proceedings of the 2025 3rd Workshop on LEO Networking and Communication*, 2025, pp. 43–49.
- [9] N. Mohan, A. E. Ferguson, H. Cech, R. Bose, P. R. Renatin, M. K. Marina, and J. Ott, "A Multifaceted Look at Starlink Performance," in *Proceedings of the ACM Web Conference 2024*, 2024, pp. 2723–2734.
- [10] D. Zhao, X. Zhang, and M. Lee, "SatPipe: Deterministic TCP Adaptation for Highly Dynamic LEO Satellite Networks," in *Proceedings of IEEE INFOCOM 2025*, pp. 1–10.
- [11] J. Garcia, S. Sundberg, and A. Brunstrom, "Characterizing the Configuration of Starlink Queuing," in *Proceedings of the 26th ACM Internet Measurement Conference*, 2026, pp. 1–10.
- [12] E. Blanton, D. V. Paxson, and M. Allman, "TCP Congestion Control," RFC 5681, Sep. 2009.
- [13] S. Floyd, J. Mahdavi, M. Mathis, and D. A. Romanow, "TCP Selective Acknowledgment Options," RFC 2018, Oct. 1996.
- [14] Y. Cheng, N. Cardwell, N. Dukkupati, and P. Jha, "The RACK-TLP Loss Detection Algorithm for TCP," RFC 8985, Feb. 2021.
- [15] F. Michel, M. Trevisan, D. Giordano, and O. Bonaventure, "A First Look at Starlink Performance," in *Proceedings of the 22nd ACM Internet Measurement Conference*, 2022, pp. 130–136.
- [16] S. Ma, Y. C. Chou, H. Zhao, L. Chen, X. Ma, and J. Liu, "Network Characteristics of LEO Satellite Constellations: A Starlink-Based Measurement from End Users," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*, pp. 1–10.
- [17] H. B. Tanveer, M. Puchol, R. Singh, A. Bianchi, and R. Nithyanand, "Making Sense of Constellations: Methodologies for Understanding Starlink's Scheduling Algorithms," in *Companion of the 19th International Conference on Emerging Networking EXperiments and Technologies*, 2023, pp. 37–43.
- [18] T. Zimmermann, E. Lanfer, D. Laniewski, S. Brinkmann, and N. Aschenbruck, "Better Fill Up Your Pipe-Revisiting Starlink's Burst Characterization," in *Proceedings of the 2025 3rd Workshop on LEO Networking and Communication*, pp. 36–42.
- [19] J. Garcia, S. Sundberg, and A. Brunstrom, "TCP Congestion Control Performance over Starlink," in *Proceedings of the 2025 Applied Networking Research Workshop*, pp. 70–77.
- [20] V. Kamel, J. Zhao, D. Li, and J. Pan, "StarQUIC: Tuning congestion control algorithms for QUIC over LEO satellite networks," in *Proceedings of the 2nd International Workshop on LEO Networking and Communication*, 2024, pp. 43–48.
- [21] Z. Lai, Z. Li, Q. Wu, H. Li, J. Li, X. Xie, Y. Li, J. Liu, and J. Wu, "LeoCC: Making Internet Congestion Control Robust to LEO Satellite Dynamics," in *Proceedings of the ACM SIGCOMM 2025 Conference*, pp. 129–146.
- [22] R. S. Frederiksen, T. G. Mulvad, I. Leyva-Mayorga, T. K. Madsen, and F. Chiariotti, "End-to-End Delivery in LEO Mega-constellations and the Reordering Problem," in *2024 IEEE 35th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pp. 1–6.
- [23] A. Morton, L. Ciavattone, G. Ramachandran, S. Shalunov, and J. Perser, "Packet Reordering Metrics," RFC 4737, Nov. 2006.
- [24] V. Jacobson, "Congestion Avoidance and Control," *ACM SIGCOMM computer communication review*, vol. 18, no. 4, pp. 314–329, 1988.
- [25] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," *ACM SIGOPS operating systems review*, vol. 42, no. 5, pp. 64–74, 2008.
- [26] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE Journal on selected Areas in communications*, vol. 13, no. 8, pp. 1465–1480, 2002.
- [27] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links," in *Proceedings of the 7th annual international conference on Mobile computing and networking*, 2001, pp. 287–297.
- [28] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-Based Congestion Control: Measuring bottleneck bandwidth and round-trip propagation time," *Queue*, vol. 14, no. 5, pp. 20–53, 2016.
- [29] N. Cardwell, I. Swett, and J. Beshay, "BBR Congestion Control," Tech. Rep., Mar. 2026, work in Progress.
- [30] W. Eddy, "Transmission Control Protocol (TCP)," RFC 9293, Aug. 2022.
- [31] J. Zhang, F. R. Yu, S. Wang, T. Huang, Z. Liu, and Y. Liu, "Load balancing in data center networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2324–2352, 2018.
- [32] 3GPP, "NR; Radio Link Control (RLC) protocol specification," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.322, 2024, version 17.4.0, Release 17.
- [33] J. V. Iyer, K. S. S. Mahammad, Y. Dandekar, R. Akella, C. Chen, P. E. Barber, and P. J. Worters, "System and method of providing a medium access control scheduler," Dec. 27 2022, uS Patent 11,540,301.
- [34] H. Cech, N. Mohan, and J. Ott, "Dissecting the StarLink: Characterizing Queuing and Flow Dynamics in the Starlink Network," in *Proceedings of the ACM SIGCOMM 2026 Conference*, 2026, pp. 1475–1495.
- [35] B. Wang, X. Zhang, S. Wang, L. Chen, J. Zhao, D. Li, and Y. Jiang, "A large-scale ipv6-based measurement of the starlink network," in *IEEE INFOCOM 2026-IEEE Conference on Computer Communications*, pp. 1–10.
- [36] F. Audet and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP," RFC 4787, Jan. 2007.
- [37] N. R. Draper and H. Smith, *Applied Regression Analysis*, 3rd ed. John Wiley & Sons, 1998, vol. 326.
- [38] N. Kuhn and E. Lochin, "Network Coding for Satellite Systems," RFC 8975, Jan. 2021.
- [39] W. Eddy, "TCP SYN Flooding Attacks and Common Mitigations," RFC 4987, Aug. 2007.
- [40] M. Handley, V. Paxson, and C. Kreibich, "Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics," in *Proceedings of the 10th USENIX Security Symposium*, 2001, pp. 115–131.
- [41] P. Srisuresh and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)," RFC 3022, Jan. 2001.
- [42] S. Karandikar, S. Kalyanaraman, P. Bagal, and B. Packer, "TCP Rate Control," *ACM SIGCOMM Computer Communication Review*, vol. 30, no. 1, pp. 45–58, 2000.
- [43] H. Balakrishnan, V. Padmanabhan, G. Fairhurst, and M. Sooriyabandara, "TCP Performance Implications of Network Path Asymmetry," RFC 3449, Dec. 2002.
- [44] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations," RFC 3135, Jun. 2001.
- [45] M. Sargent, J. Chu, D. V. Paxson, and M. Allman, "Computing TCP's Retransmission Timer," RFC 6298, Jun. 2011.
- [46] J. Lim, M. Choi, M. Park, and J. Paek, "RedHAT: Receiver-driven Handover Adaptation in TCP for Starlink Communication," in *The 17th International Conference on Information and Communication Technology Convergence (ICTC)-Workshop on Big Data, CPS, and 5G&6G Communication Networks (IWBCN)*, Oct. 2026.
- [47] H. Kim, G. Lee, J. Shin, J. Paek, and S. Bahk, "Slot-Size Adaptation and Utility-Based Packet Aggregation for IEEE 802.15. 4e Time-Slotted Communication Networks," *IEEE Internet of Things Journal*, vol. 11, no. 9, pp. 16 382–16 397, 2024.
- [48] —, "Quick6TiSCH: Accelerating formation of 6TiSCH networks with TSCH and RPL," in *IEEE 21st International Conference on Mobile Ad-Hoc and Smart Systems (MASS)*, 2024, pp. 66–74.
- [49] P. Thubert, "IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Selective Fragment Recovery," RFC 8931, Nov. 2020.