SATPIPE: Deterministic TCP Adaptation for Highly Dynamic LEO Satellite Networks

Ding Zhao*, Xinyu Zhang†, Myungjin Lee‡
Johns Hopkins University*, University of California San Diego†, Cisco Research‡
Emails: dzhao29@jhu.edu*, xyzhang@ucsd.edu†, myungjle@cisco.com‡

***Abstract*—Low-Earth-Orbit (LEO) satellite networks (satnets) hold significant potential for providing global internet access with high throughput and low latency. However, the high mobility of the satellites and the associated handovers cause high dynamics at the link layer, which in turn degrades end-to-end network throughput and stability. In this work, we first present a measurement profiling of the handover behaviors of the Starlink satnet, so as to better understand its dynamics and accurately determine handover timings with millisecond precision. We then introduce SATPIPE, a new mechanism to enhance TCP and make it robust against satnet dynamics. SATPIPE exposes the link layer handover schedule to the TCP sender, which then adapts to the link interruptions in a deterministic rather than trial-and-error manner. Our implementation and experiments over Starlink indicate that SATPIPE delivers an average throughput gain of 9.4% to 38%, achieves up to a 127.8% enhancement in the 10% lower throughput, and exhibits a 24.7% reduction in retransmission ratio, compared to the state-of-the-art TCP BBR. This advantage further propagates to the application layer, leading to a 10.8% increase in bitrate and 33.5% reduction in rebuffering time for video streaming applications.**

I. INTRODUCTION

Low Earth Orbit satellite networks (satnets) are emerging as a transformative technology for internet connectivity, offering low-latency and high-throughput access at planet scale. Recent LEO mega-constellations such as Starlink [1], Kuiper [2] and OneWeb [3] are poised to enable last-mile network connectivity by integrating with existing terrestrial network infrastructures. Among these, Starlink has taken the lead with over 5000 operational satellites [4], achieving 100 Mbps to 200 Mbps throughput with latencies as low as 20 ms [1].

Unlike conventional geostationary (GEO) satellites, LEO satnet constellations orbit the Earth at high velocities to counteract gravitational forces. This intrinsic characteristic imposes an unprecedented challenge for LEO satnets, *i.e.*, the link dynamics. To ensure continuous connectivity towards ground-based user terminals, the serving satellite needs to perform frequent handovers. Although each handover only lasts a few tens of *ms*, this short "freeze" of the satnet link often leads to a burst of packet losses and RTT escalation [5]–[8],. The performance degradation tends to be amplified at the higher layers. In particular, the transport layer protocols (*e.g.*, TCP CUBIC [9] and BBR [10]) rely on the loss/delay estimation as indicators of network congestion. BBR reacts slowly since it relies on smoothed measurements across multiple RTTs; and when BBR actually respond, it tends to overreact, deeming the burst of delay/loss as equivalent to a severe network congestion. Consequently, TCP tends to underutilize the satnet capacity even in an ideal situation without any congestion. Although such handover-induced instability also occurs in other mobile networks such as cellular networks, the impact is less pronounced due to shorter RTTs and hence faster recovery times [11].

In this paper, we first conduct a cross-layer measurement study to unravel the impacts of satnet handover on TCP performance. Our measurement begins by examining the physical layer behaviors including signal strength patterns under satellite mobility, using Starlink as a reference system. Then, we investigate the link-level behaviors at fine-grained millisecond timescale, from which we derive the precise handover starting/ending times, frequency and duration. Our measurement results corroborate the hypothesis in prior works [12] and the Starlink's FCC filing [13], which stipulate that a new satellite has to be scheduled for the user terminal around every 15 s. Such periodicity will likely prevail in other LEO satnets because the satellites are usually uniformly spread out within the constellation. Furthermore, our measurement reveals that, each handover lasts around 50 ms, yet the TCP's response lags behind by a few hundred ms, and TCP's performance degradation often lasts a few seconds before it recovers the congestion window close to the true network capacity.

Based on the measurement observations, we propose a new mechanism called SATPIPE to enhance TCP performance over dynamic LEO satnets. SATPIPE follows two key design principles to tailor the TCP congestion control: *(i) Visibility:* SATPIPE exposes the handover-induced link-level dynamics to the transport layer, rather than relying on the trial-and-error adaptation based on end-to-end delay/loss statistics. *(ii) Determinism:* SATPIPE leverages knowledge of the periodic handover events to plan its reactions in a deterministic manner. Ideally, these two principles should together bring TCP sending rate close to the "ground-truth" capacity of the satnet link. However, practical implementation of SATPIPE begs two non-trivial questions.

(i) *How does* SATPIPE *know the handover time?* Our observations of practical satnets reveal that simply using short-term packet loss or RTT statistics cannot discriminate satnet handover from ordinary network capacity variations (*e.g.*, due to congestion in the backbone network). We thus propose two solution mechanisms. The first employs the network time protocol (NTP) to synchronize the TCP sender to the global schedule of handover, which is known for a given satnet system through the "two-line element" [12]. This solution requires the network nodes to periodically probe the NTP server. Our second solution further evades this overhead, by using a time-domain periodicity detection algorithm, which

accumulates loss/RTT evidence across multiple handover periods in order to pinpoint the periodic timestamps when the network is "interrupted".

(ii) *How should* SATPIPE *react to a handover event, so as to maximize utilization of the end-to-end network capacity while maintaining fairness?* To match the congestion window to the available network capacity, SATPIPE leverages BBR [10], a state-of-the-art rate-based congestion control protocol that employs delivery rate and delay together to estimate the bandwidth-delay product. To overcome BBR's slow reactions, SATPIPE updates its state machine, and introduces a *queue maintenance* state that explicitly flushes the data build-up due to the short-term network freezing during handover, forcing the end-to-end network path to accurately estimate the RTT by isolating the interference from queuing delay. Unlike generic TCP protocols, SATPIPE explicitly and deterministically respond to short-term network capacity due to handovers. Unlike recently proposed link-layer aware TCP adaptation [12] that directly inhibits the congestion control during handover, SATPIPE is essentially still a reactive protocol. It approximates the optimal congestion window through an accelerated estimation rather than blind inhibition, thus maintaining both efficiency and fairness to competing flows.

We have implemented SATPIPE as a lightweight driver patch to state-of-the-art TCP BBR. Furthermore, we have conducted comprehensive experiments, covering diverse network conditions including different path lengths, background traffic levels, etc. We found that SATPIPE delivers an average throughput gain of 9.4% to 38% compared with BBR. Zooming in the 10% percentile, SATPIPE achieves 127.8% throughput enhancement. We have also evaluated SATPIPE in a Dynamic Adaptive Streaming over HTTP (DASH) video streaming system, which demonstrates 10.8% improvement in average bit-rate and 33.5% reduction in rebuffering time.

In summary, our work makes the following contributions:

- We conduct fine-grained measurements to characterize link handover and outage timing of real-world LEO satnets.
- We design SATPIPE, the first TCP adaptation algorithm that reacts to handover interruptions in an explicit and deterministic manner.
- We implement SATPIPE as a patch to the standard TCP module and validate its performance in Starlink satnet.

II. RELATED WORK

Starlink PHY layer signal identification and localization. Recent measurement studies have investigated the physical layer signal characteristics of Starlink. Researchers [14], [15] have developed blind signal identification techniques to understand detailed signal structure and to simulate received signals for positioning applications. Based on the signal structure and prior knowledge, the beamforming strategies of Starlink satellites were studied in [16]. A Ku-band dish antenna was used to capture the Starlink signals, combined with a low-noise-block (LNB) for converting the signals to lower frequencies and a USRP software radio for baseband processing. A similar setup has been used to repurpose Starlink downlink tones for opportunistic positioning [17], [18].

Starlink upper layer characterization and evaluation. Existing research has shown that Starlink can achieve much lower latency and higher throughput compared to conventional satellite communications using GEO satellites [5]. However, significant packet losses occur during satellite handovers, as highlighted in [6]. These packet losses lead to substantial throughput reduction on the transport layer. To gain insights into the timing details of the Starlink system, multi-timescale measurements were conducted at the network layer [19]. These measurements identify a coarse-grained handover period of 15 s at medium timescales and observe decreased link utilization during handovers. The scheduling mechanism of Starlink satellites is comprehensively studied in [7], which confirm the allocation period and reveal that local on-satellite controllers manage flow scheduling from user terminals. In this paper, we use inter-packet delay (IPD) to estimate the exact time of handovers, rather than a metric for network stability.

TCP adaptation in dynamic networks. Cross-layer congestion control has been widely explored in cellular networks to address the mobility and resource allocation problems. In particular, leveraging physical and link layer statistics to inform TCP adaptation shows promise in dynamic cellular networks. For instance, CLAW [11] and piStream [20] use signal strength information to predict available bandwidth of the cellular last-hop link. This prediction is in turned used to drive precise rate adaptation at the transport layer or application layer. SATCP [12] is a link-layer informed TCP adaptation algorithm for LEO satnets, which shares similar spirit as SATPIPE, and was evaluated using the LeoEM emulator. SATCP assumes the ground station can provide explicit handover timestamp to the user terminal to assist the end-to-end TCP rate adaptation. Due to lack of information about the handover duration, SATCP conservatively freezes the congestion window (CWND) for 2.3 seconds, which results in suboptimal use of available network capacity and potential fairness issues. In contrast, SATPIPE is an end-to-end mechanism running entirely on the TCP hosts. Our experiments within the operational Starlink satnet verify its superior performance over SATCP.

III. MEASUREMENT STUDY OF STARLINK HANDOVER

In this section, we present comprehensive measurements of Starlink handover behaviors at both the physical layer and network layer, uncovering the fundamental mechanisms of the Starlink global controller.

A. PHY Layer Measurements

The Starlink downlink signal features 8 channels, each occupying a bandwidth of 240 MHz within the Ku-band (10.7 GHz to 12.7 GHz) [14], [21], with a 10 MHz guard band between adjacent channels. Such large bandwidth exceeds the capabilities of most commercial software-defined radios (SDRs). However, in the middle of each active channel, there are multiple tones occupying several MHz of bandwidth,

which can be treated as the indicators of the Starlink downlink signals.

We have established a testbed setup to capture these downlink tones following recent work [18], as illustrated in Fig. 1. Using a universal Ku-band LNB [22], we capture and down-convert the Starlink signals from the Ku-band to lower frequencies. While a parabolic dish could enhance received signal strength, we opted for the LNB-only approach to widen the field of view, facilitating signal reception from multiple satellites. The LNB connects to a USRP N310 [23] for baseband signal processing. The N310 samples at 2.5 Msps with and centers at 11.325 GHz – the carrier frequency of one of the active Starlink downlink channels. To ensure the downlink transmission is active, we continuously download high-definition videos through the Starlink User Terminal (UT), which is co-located with the LNB during measurements.

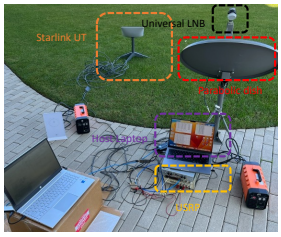


Figure 1. Starlink downlink physical layer signal receiver

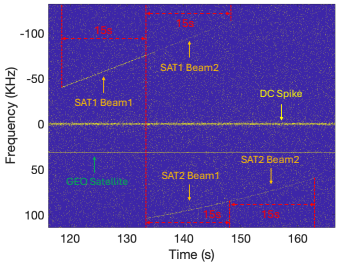


Figure 2. Received Starlink downlink middle channel signals.

Fig. 2 shows the frequency map of different satellites, including broadcast GEO satellites and Starlink LEO satellites. We can observe two different Starlink satellites based on Doppler shifts. Satellite 1 exhibited the strongest signal strength for 15 seconds before experiencing a sharp drop, indicating a handover event. Subsequently, Satellite 1 switched to a different beam (beam 2), possibly covering a different region, so the leaked signal strength captured by our receiver became much weaker. Meanwhile, Satellite 2 showed the strongest signal strength. After 15 seconds, Satellite 2 also switched beams, but no further handovers were detected. The reason for this could be that the satellite used an alternate active channel for data transmission, while our signal detection system is set up to monitor only a single channel. We conducted multiple independent experiments and the measurement results consistently support our findings and speculations. Based on the physical layer observations, we can conclude that each satellite indeed serves a target region for approximately 15 seconds before it hands over to the next.

B. Fine-grained Handover Profiling Using Network Statistics

For a more fine-grained understanding of Starlink handovers, we measure the packet-level RTT and TCP throughput variations. We generate TCP traffic from an AWS EC2 server iPerf3 3.14, and receive the data using a Starlink UT. The UT is located in California, whereas the server is in Ohio by default. A 1 Gbps Ethernet cable connects the UT with a PC which acts as the TCP client.

We repeat several iperf3 measurements running with single TCP connection using CUBIC. We then calculate the throughput within each 100-millisecond window. The results in Fig. 3 reveal significant TCP throughput drops around every 15 seconds. Fig. 4 further shows the periodic pattern of RTT, which is likely caused by the routing path changes upon handover from one satellite to the next. We also notice that while RTT variance can indicate periodic handovers, there are times when changes in RTT are not significant, which makes it difficult to pinpoint the exact timing of the handover directly from RTT.

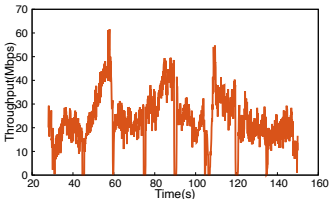


Figure 3. Throughput measurement from OH to SD using TCP CUBIC shows periodic throughput drop.

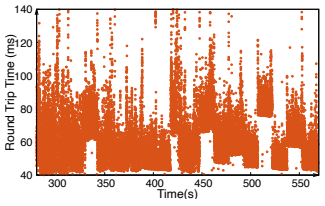


Figure 4. RTT collected in TCP BBR shows periodic patterns.

Meanwhile, after analyzing the absolute timing of each received packet, we found the RTT change tends to occur deterministically, at the 12th, 27th, 42nd, and 57th second past every minute. This observation matches recent measurement studies [7].

To obtain a precise estimation of the handover duration, we establish multiple independent TCP connections from the client to the AWS EC2 server and use TCPdump [24] to capture the timestamp of the receiving packets. We record the timestamps of two consecutively received packets and obtain the corresponding IPD. *The IPD should be a random variable with a negligible mean value when the satellite handover is not included between the two consecutive packets, and should have a relative large mean value otherwise.*

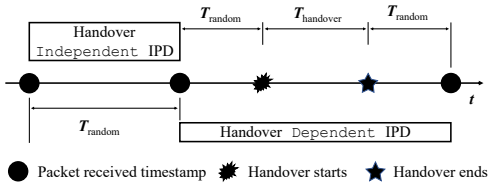
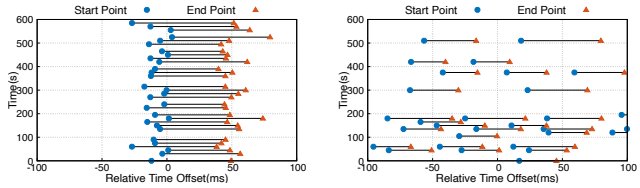


Figure 5. Illustration of calculating inter-packet delays.

We perform a 10-minute iperf3 measurement and analyze received packets whose the timestamps falls within the intervals [12s-Δ, 12s+Δ], [27s-Δ, 27s+Δ], [42s-Δ, 42s+Δ], [57s-Δ, 57s+Δ] past every minute. The value of Δ is set to 100 ms, used as an estimated upperbound for handover duration. Having a coarse-grained estimate of when the handover happens decreases the amount of data we need to process, and mitigates the interference from backbone traffic. These data points constitute the experimental group, from which the IPD values are extracted. In order to more accurately depict the variance in IPD distributions in different time slots, we use the intervals [10s-Δ, 10s+Δ], [25s-Δ, 25s+Δ], [40s-Δ, 40s+Δ], [55s-Δ, 55s+Δ] as control group, which deviates from the ground-truth

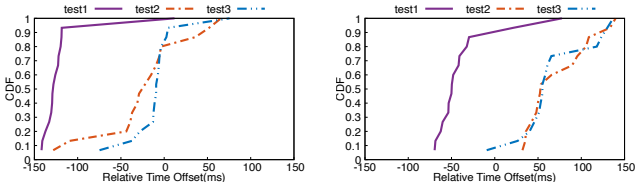
handover timing (observed in the physical layer measurement) by around 2 seconds. Since handovers occur about every 15 seconds, we expect at most 40 handovers in a 10-minute test. To minimize the possible IPD outliers due to backbone network congestion, we take the largest 30 IPDs out of all the data points within the experimental and control groups, respectively. We then remap the timestamps to the [0s, 15s] period and plot the IPD values in Fig. 6. Each line segment represents an IPD, with IPDs from the experimental group serving as unbiased estimations of handover duration.The time offset of the experimental group is relative to 12s past every minutes, while the control group is relative to 10s.



(a) Experimental group (b) Control group

Figure 6. Distribution and length of Top 30 IPD values

Unlike randomly distributed IPDs in the control group, the top-30 IPDs in the experimental group are densely concentrated within the same interval, around [-30,60] ms. These periodic large IPDs are caused by the link outage during the handover. These IPDs can be considered as unbiased estimates of handovers, allowing us to deduce that the estimated handover duration is almost always less than 100 ms.



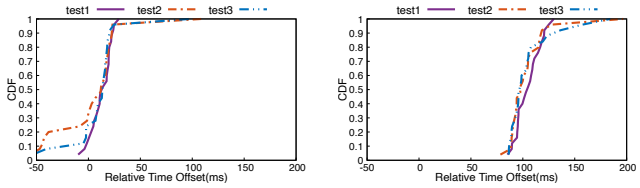
(a) CDF for IPD starting time (b) CDF for IPD ending time

Figure 7. CDF for IPD starting and ending time before NTP synchronization

However, while the duration of the link outage remains relatively stable, the starting and ending times vary across measurements. We conduct these measurements from different servers to different clients. The CDF for the starting and ending times of each measurement is depicted in Fig. 7. Clearly, the relative starting and ending times differ across various measurements, where the timestamps are generated by different local clocks. Therefore, time synchronization plays a crucial role in reducing the estimation error of the handovers for different servers and clients.

To verify the feasibility of truly deterministic estimation of the handover timing, we synchronize the client to a global stable clock through Network Time Protocol (NTP) [25]. Fig. 8 illustrates the CDF for both starting and ending times, which are highly consistent across different measurement tests.

In summary, we can conclude that the handover process in Starlink follows a periodic and deterministic pattern. All the handover events are observed to take place within some specific time intervals: [12s + Δ_1, 12s + Δ_2], [27s + Δ_1, 27s + Δ_2], [42s + Δ_1, 42s + Δ_2], [57s + Δ_1, 57s + Δ_2]. Using a conservative setting of $\Delta_1 = 15\ ms$ and $\Delta_2 = 100\ ms$, we observe no outliers.



(a) CDF for IPD starting time (b) CDF for IPD ending time

Figure 8. CDF for IPD starting and ending time after NTP synchronization

IV. SatPipe System Design

In this section, we explain the system design logic behind SatPipe and how it can be implemented through a lightweight patch to existing TCP.

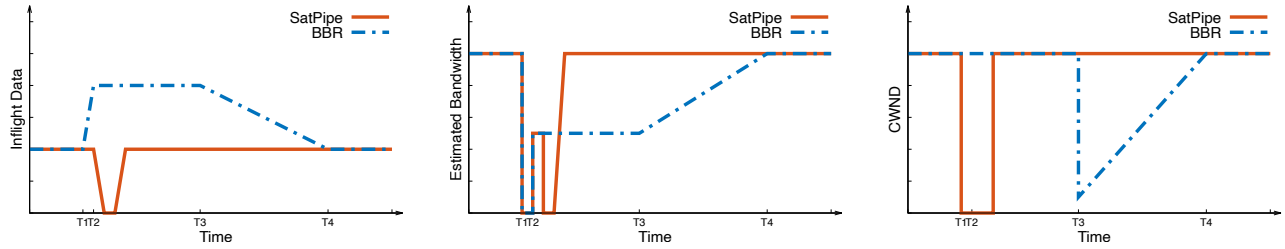
A. SatPipe *Overview*

SatPipe is a generic TCP enhancement mechanism that mitigates the inaccurate responses of TCP to handover-induced network dynamics. SatPipe employs two methods to determine when a handover occurs: an NTP-based scheme, requiring the TCP sender to synchronize to global clock, thus knowing the *absolute timing* of periodic handover events; and a blind estimation scheme, requiring an initial IPD measurement period (around 3 minutes) to determine the *relative timing* of handovers, relative to the sender's local clock, with periodic recalibration of clock drift in the background.

Given the handover schedule, SatPipe modifies the TCP behavior during each handover period. It freezes classical TCP's short-term overreaction to the RTT/loss surge during handover. Instead, it enforces a reestimation of the available network capacity and reapplies the TCP's CWND estimation method immediately after the handover period. This design principle is applicable to all TCP variants. Yet without loss of generality, we demonstrate its use for the state-of-the-art TCP BBR. BBR is preferred over the widely used CUBIC due to satellite link instability because Starlink experiences persistent packet loss, and CUBIC's sensitivity to packet loss impedes reaching bottleneck bandwidth efficiently. In contrast, BBR can utilize available bandwidth more efficiently than CUBIC, as we will show later in our experimental validation.

B. *How does TCP BBR React to Satnet Handover?*

Before introducing the detailed design of SatPipe, we explain why BBR struggles to fully utilize the bandwidth in satnets. To this end, we compare its behavior when running across a WiFi network versus the satnet. Specifically, we first rerun the iperf3 TCP measurement using aforementioned AWS EC2 server and Starlink user terminal connection. Then we change the last-hop to be a campus WiFi network. Our experiments ran during late evenings to mitigate the interference from competing traffic. Fig. 10 plots the measured time-series



(a) In-flight Data (b) Estimated Bandwidth (c) CWND

Figure 9. TCP metrics during transmission

of BBR throughput in both cases. We observe that the BBR throughput over WiFi is relatively stable, except for a sharp, ephemeral drop-and-recovery around every 10 seconds. On the other hand, when running over the Starlink satnet, BBR experiences prolonged periods of throughput degradations. We also ran a UDP session which reveals stable network capacity with periodic, ephemeral drop every 15 seconds. Yet the TCP BBR throughput is much lower than this upper limit. BBR's behavior also drastically differs from other TCP variants such as CUBIC (Fig. 3), which shows a clear periodic throughput drop corresponding to the 15 second satnet handover schedule.

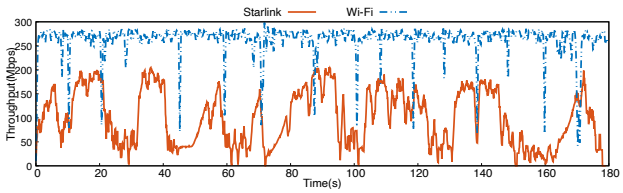


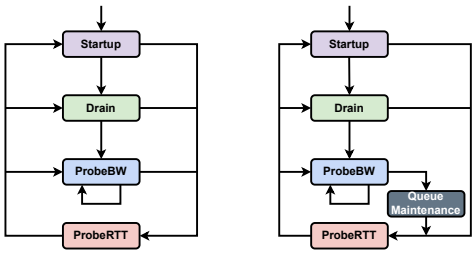
Figure 10. BRR throughput variance over Starlink network and Wi-Fi

To investigate the reasons behind such discrepancies, we visualize the BBR behavior and corresponding transport layer statistics before/after handover in Fig. 9. When the handover starts (at timestamp T1), the delivery rate abruptly drops to zero due to last-hop link outage. However, the CWND remains unchanged across the short handover duration, because BBR uses bottleneck bandwidth (BtlBW) filter to estimate the bandwidth, resulting in a moving-average lagging effect. Consequently, the TCP sender keeps injecting a large amount of data into the end-to-end network pipe. Upon completion of the handover and a new satnet link being reestablished (at T2), the in-flight data exceeds the bandwidth delay product (BDP) for a period of time, resulting in a surge in RTT, which in turn causes a drop in the estimated bottleneck bandwidth. Once the bandwidth filter in BBR expires (at T3), the CWND is reduced to a relative low value in commensurate with the low estimated bandwidth, and it requires a significant amount of time to recover to its normal level (T4) due to the slow additive increase.

We note that the WiFi network does not experience a long-lasting throughput degradation (Fig. 10), mainly because the network capacity is relatively stable and there is no drastic in-flight queue buildup. The periodic sharp throughput drop (around every 10 s) is mainly due to the RTT probing mechanism in BBR, which attempts to flush in-flight data to obtain an accurate estimation of RTT by minimizing queue effects (Section IV-C).

C. SATPIPE *Workflow*

Fig. 11 illustrates the state machine of SATPIPE in contrast to BBR. SATPIPE introduces an additional stage called `Queue Maintenance` to manage CWND during handovers. Otherwise, it remains similar to the default BBR as detailed below:



(a) BBR workflow (b) SATPIPE workflow

Figure 11. State machines for BBR and SATPIPE

`Startup`: Similar to the slow start phase of classical TCP, this stage exponentially increases the sending rate and CWND to probe the current bottleneck of the network. The growth rate of CWND is precisely designed so that it will compete fairly with other data flows.

`Drain`: If the RTT is increasing while the estimated bandwidth remains essentially unchanged, BBR determines the system has reached its bottleneck capacity. Accordingly, it enters the `Drain` phase, which rapidly changes the sending rate to reduce the in-flight data volume to the size of the BDP.

`ProbeBW`: Once the in-flight data volume matches the BDP, the system transitions to the `ProbeBW` phase, where it spends most of its time. Pacing gain adjustments (1.25, 0.75, 1, 1, 1, 1, 1, 1) intermittently increase the sending rate to check for increased bottleneck bandwidth.

`ProbeRTT`: In this stage, BBR sets CWND to a very small value to drain the current amount of in-flight data and ensures that the RTT is a theoretical minimum during this time to adapt dramatically environment change. BBR enters `ProbeRTT` once the estimation of round-trip propagation (RTprop) has not been updated for 10 seconds by default.

`Queue Maintenance`: During the predefined handover intervals, SATPIPE forces the stage machine to transition

into the `Queue Maintenance` phase. Here, CWND is reduced to 0 to prevent queue buildup during handovers. After a short period (equal to the handover duration) in `Queue Maintenance`, SATPIPE transits directly to `ProbeRTT`. This step is crucial as it prevents network queue overflow during handovers and forces an instant update of the propagation delay estimation to adapt to new routing conditions, rather than waiting for the default 10-second RTprop update as in BBR.

The change in transport layer statistics for SATPIPE is illustrated in Fig. 9. During handover, SATPIPE automatically reduces the CWND, preventing an increase in in-flight data. After `ProbeRTT`, SATPIPE quickly restores the original CWND. During handover, the estimated bandwidth falls to zero as no ACKs are received. When the handover is complete, the previously buffered ACKs, delayed by the handover, reach the sender and cause a relatively low estimated bandwidth, similar to BBR. Subsequently, because of the low CWND during `ProbeRTT`, the estimated bandwidth stays at zero until the CWND returns to original level.

D. Other Design Considerations

Eliminating dependency on external handover indicators. SATCP [12]–state-of-the-art satnet TCP design–mainly relies on incoming handover reports from ground stations. A relay application is designed in the user space to listen to handover reports, pass them to the kernel, which further triggers a freezing of the CWND during the handover period. However, such external report signals can introduce unnecessary delays and overhead that require a longer freezing window, thereby increasing the likelihood of using an outdated frozen CWND, which either causes underutilization of network capacity or unfair aggression over competing flows.

In contrast, SATPIPE can abandon the handover report design thanks to its lightweight mechanism for accurate handover time estimation (Section IV-E). We implement the handover estimation directly in the kernel to ensure instantaneous response to handovers.

CWND inhibition versus reactive CWND adaptation. SATCP's proactive approach of inhibiting the CWND may inadvertently affect TCP fairness. Despite the inhibition window being relatively short, maintaining the same CWND before, during, and after the handover may still unfairly deprive bandwidth from competing flows in the backbone network, especially those newly started flows during the handover period. In effect, SATCP's default 2.3-second CWND inhibition is equivalent to tens of RTTs which amplify the effect.

SATPIPE represents a reactive approach for TCP adaptation, focusing on reducing the sending rate during handover and flushing the queue afterwards. Thanks to the filter in the BBR protocol, the estimated available bandwidth remains unaffected. Draining the queue for 100 ms results in a 1% decrease in throughput performance but we can gain a lot more by preventing CWND fallback due to queue buildup.

E. Blind Estimation of Handover Time

NTP is the default "time server" for most major Linux distributions [25]. As mentioned in Section III-B, when both ends of the TCP are synchronized to NTP, the handover schedule can be predicted in an almost deterministic manner. Frequent NTP synchronization through the satellite link may affect the throughput performance. Fortunately, NTP dynamically chooses the optimal poll interval within the range defined by a `minpoll` and `maxpoll`, which typically default to 64 and 1024 seconds [25], respectively. These values are suitable for the majority of environments and the clock mismatch is negligible within the polling intervals. Given that these polling intervals are relatively long compared to the handover window we intend to operate, the NTP synchronization overhead can be ignored for SATPIPE.

To address the cases when the NTP synchronization is unavailable (*e.g.*, due to security restrictions) to the TCP nodes, we introduce a blind estimation algorithm. Specifically, we estimate the precise handover time by analyzing the IPDs. By mapping the start and end times of each IPD to a [0,15] second interval, we create a counter for each small time slot (1 ms) to represent the possible intervals in which the large IPDs occur. Given the deterministic nature of handover times and the random distribution of non-handover related IPDs, a sufficient numbers of IPD samples will result in a peak in the counter corresponding to the handover time, as depicted in Fig. 12. Identifying the peak location allows us to pinpoint the precise handover time.

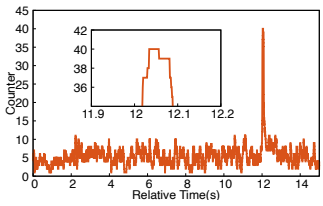


Figure 12. Starlink handover blind estimation empirical distribution

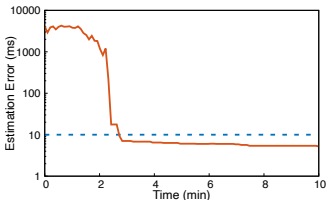


Figure 13. Blind estimation error compared to NTP synchronization

The pseudo-code for the blind estimation is shown in Algorithm 1, where `IPD` is the difference between adjacent received packet timestamps and `topIndex` is the list containing the index of top `num` largest IPDs. The blind estimation can be executed in user space, and the results can be passed to the kernel for handover time synchronization without NTP.

Theoretically, the more IPD samples we collect, the higher confidence the blind estimation can achieve. Fig. 13 shows the estimation error relative to the NTP ground-truth, as we accumulate more IPD samples. We observe that achieving an error of less than 10ms requires capturing 3 minutes of IPD samples. This implies that the user terminal (or the client device connected to the user terminal) needs to initialize itself by running the 3-minute blind estimation once in a bootstrap phase. Afterwards, it only needs to rerun the calibration periodically to offset potential local clock drift. Such recalibration can be performed in the background because it only requires collecting and analyzing the IPD samples. Modern computing devices commonly have a clock stability of 10ppm or less, corresponding to at most 18 ms of drift over 30 minutes.

Given that the handover during is around 100 ms, such drift is unlikely to adversely impact the SATPIPE operations. In other words, recalibration in the background for every 30 minutes should suffice.

Algorithm 1: Blind Estimation of Handover Time

```
Data: PktTimestamp: List of Int
Result: peakPosition: Int
Res ← 15000 // Millisecond resolution
Cnt ← zeros(Res) // Create Counter list
IPD ← diff(PktTimestamp)
topIndex ← argsort(IPD, num)
n ← 0
while n <= len(topIndex)-1 do
    index ← topIndex[n]
    start ← int(PktTimestamp[n])%Res
    end ← int(PktTimestamp[n+1])%Res
    if start < end then
        for i ← start to end do
            Cnt[i] ← Cnt[i] + 1
    else
        for i ← start to Res-1 do
            Cnt[i] ← Cnt[i] + 1
        for i ← 0 to end do
            Cnt[i] ← Cnt[i] + 1
    n ← n + 1
peakPosition ← argmax(Cnt)
return peakPosition
```

F. SATPIPE Implementation

SATPIPE is implemented as a lightweight kernel patch to the TCP driver. The pseudo-code for SATPIPE is shown in Algorithm 2. We emphasize that SATPIPE runs entirely at the *TCP sender side*. The aforementioned blind estimation uses the IPD of consecutive ACKs to gauge handover schedule, which in turn drives the state machine in Algorithm 2.

The boolean variable `Queue_Maintenance_Flag` serves as an indicator to determine whether the congestion window should currently be set to zero to avoid queue stacking during the handover, and `Force_RTT_Flag` determines whether SATPIPE should transition to the `ProbeRTT` phase to drain the in-flight data and obtain accurate RTprop estimation.

In `ProbeRTT`, the CWND is set to 4 maximum segment size (MSS) and lasts for 200 ms by default. Given the short handover duration and the requirement for `ProbeRTT` immediately following the handover, we can merge `ProbeRTT` and `Queue Maintenance` by prolonging the `ProbeRTT` duration. Similar to `Queue Maintenance`, `ProbeRTT` also sets CWND to a low value to make sure the queue is empty and measures the RTT. This adjustment allows the system to enter `ProbeRTT` when the handover starts. Based on previous measurements, the `ProbeRTT` duration can be set to (200 + handover duration) milliseconds.

It is important to note that the SATPIPE is not strictly hard-coded to the Starlink characteristics. The periodicity and determinism of handover is a universal phenomenon due to the periodic nature of the LEO satellite constellation. SATPIPE's blind handover estimation algorithm ensures that it can adapt to any LEO satnet's deterministic handover schedule.

Algorithm 2: SATPIPE implementation on Linux

```
Procedure Queue_Maintenance()
    if Current_Time ∈ [T_start, T_end] then
        CWND ← 0
        Force_RTT_flag ← True
        Queue_Maintance_flag ← True
    else
        Queue_Maintance_flag ← False
Procedure Force_RTT_Probing()
    if Force_RTT_flag and
     !Queue_Maintance_flag then
        Force_RTT_flag ← False
        Enter ProbeRTT()
```

V. EVALUATION

In this section, we evaluate SATPIPE in comparison with widely deployed TCP protocols including Reno, CUBIC [9], the state-of-art BBR [10] and SATCP [12] across a wide range of scenarios in real-world satnets. We also examine how well SATPIPE's performance gain translates into improved quality of experience for video streaming applications.

A. Experimental Setup

We performed our measurements using Gen-2 Starlink user terminal, installed in open areas. The user terminal connects to the laptop via a 1Gbit Ethernet cable instead of WiFi link, which avoids the uncertainties caused by the wireless links.

We conducted experiments using 6 AWS EC2 server instances located in different places, each representing different path lengths: North California (NC), Oregon (OR), Ohio (OH), London (LD), Singapore (SG) and Canada (CA). The EC2 servers run Amazon Linux 2023 with kernel version 6.8.10. We used a Linux laptop as the client, located in San Diego, California. By default, the clients and servers are NTP synchronized.

For application-level measurements, we modified dash.js [26] for DASH video streaming to collect enhanced video metrics including bitrate and rebuffering time. In our assessments, we used Big Buck Bunny [27] as the reference video, encoding it with the H.264/MPEG-4 standard. This video is encoded at bitrates in {210, 951, 1451, 2642, 4906, 7427, 13611} kbps, pertaining to video resolutions of {144, 360, 480, 720, 1080, 1440, 2160}p. The video has a total length over 10 minutes, and is divided into 318 chunks following the DASH standard. Unless otherwise noted, DASH servers ran on Amazon EC2 t2.large instances.

Similar to the state-of-the-art video streaming system [28],

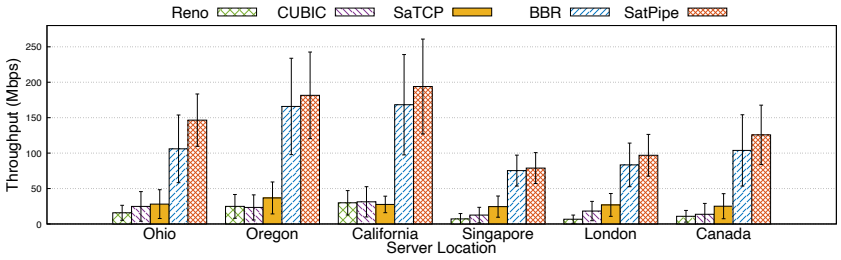


Figure 14. Comparing SATPIPE with existing TCP protocols by analyzing their performance through iperf3.

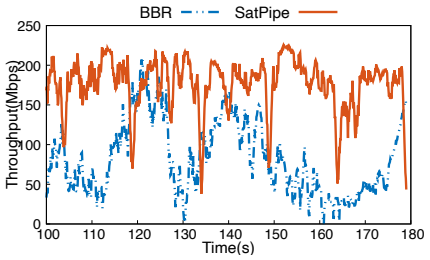


Figure 15. Starlink throughput measurement using iperf3 shows SATPIPE judiciously reduces CWND during handover and recover rapidly.

we consider a variety of QoE metrics derived from the following general equation with different parameters:

$$QoE = \frac{1}{N}(\sum_{n=1}^{N} q(R_n) - \lambda \sum_{n=1}^{N-1} T_n - \mu \sum_{n=1}^{N-1} |q(R_{n+1}) - q(R_n)|)$$

for a video with N chunks. R_n denotes the bitrate of chunk n, and $q(R_n)$ translates this bitrate into the perceived quality by a user. T_n indicates the rebuffering time caused by fetching chunk n at bitrate R_n. λ and μ represent the penalty for rebuffering time and smoothness, respectively.

We assess three options for $q(R_n)$, including linear bitrate utility used by MPC [29], logarithmic utility used by BOLA [30] and High Definition (HD) favored utility introduced in Pensieve [28]. We also examine SATPIPE across various QoE preferences, including balance, smoothness preference (SP), and favoring a low stall rate (RP). The parameters for different QoE metrics are summarized in Table I.

To minimize the effect of network traffic variations throughout the day, all the measurements are carried out consecutively in the shortest possible time frame. Also, we intentionally introduce additional background traffic to represent real-world scenarios with competing flows.

Table I
THE QOE PARAMETERS WE CONSIDER IN OUR EVALUATION.

	QoE linear	QoE log	QoE HD
$q(R_n)$	R_n	$log(R_n/R_{min})$	$R_n log(R_n/R_{min})$
$\lambda_{balance}$	R_{max}	$log(R_{max})$	R_{max}
λ_{RP}	$2R_{max}$	$log(2R_{max})$	$2R_{max}$
$\mu_{balance}$	1	1	1
μ_{SP}	2	2	2

B. Performance Evaluation

SATPIPE throughput and retransmission ratio. We performed multiple 3-minute iperf3 measurements with single TCP connection from the EC2 servers to the Starlink UT. Fig. 14 shows the average throughput over Starlink for servers at different locations. Error bars span ± one standard deviation from the mean unless otherwise noted. We notice that packet loss based algorithms, such as TCP Reno, TCP CUBIC and SATCP can not utilize available bandwidth efficiently. For example, BBR can achieve 4.3× to 7.6× the average throughput of CUBIC. One possible reason is that loss-based algorithms are overly conservative in responding to frequent losses over the satellite wireless link. SATCP keeps the CWND unchanged before and after the handover, thus experiencing a performance boost compared to CUBIC in most cases. However, SATCP's aggressive congestion inhibition window (2.3s) may result in a prolonged period of suboptimal CWND, causing only minor improvement or even performance degradation. Also, packet loss on the wireless link during non-handover periods is the main factor behind SATCP's inferior performance compared to BBR in real-world measurements, as LeoEM [12] does not account for wireless link characteristics.

In contrast, SATPIPE can increase average throughput by 9.4% (SD ↔ SG) to 38.2% (SD ↔ OH) compared to BBR. For instance, SATPIPE can achieve 200 Mbps throughput over the SD ↔ NC path, which reaches the limit of the link capacity claimed by Starlink [1]. Zooming in the long-tails, we observe that SATPIPE significantly enhances the throughput performance at lower percentiles. Specifically, SATPIPE increases throughput at the 10th percentile by up to 127.8% (SD ↔ OH), effectively addressing the issue of transient throughput drops during handover periods [12].

SATPIPE is designed to mitigate the effect of bursty queue accumulation during the handover. SATPIPE reduces CWND at a particular time interval every 15 seconds and the throughput should recover rapidly after the handover. Fig. 15 demonstrates that the real-world throughput performance of the Starlink network aligns with the design of our system. Additionally, the throughput behavior is similar to the ideal behavior of BBR observed in a stable WiFi network as shown in Fig. 10.

We further compare the traffic patterns of BBR and SATPIPE from Ohio to California. BBR takes a considerable amount of time to recover from the effects of the handover, which can negatively impact real-time applications, such as video streaming. In contrast, SATPIPE ensures that there is no queue buildup thanks to properly designed `Queue_Maintenance` and `ProbeRTT` stages, thus maintaining high CWND after the handover. The periodic CWND drop for SATPIPE in the figure is caused by `Queue_Maintenance` and `ProbeRTT`, lasting for around 300ms, during which SATPIPE tries to flush the in-flight data accumulated during the handover.

Fig. 16 shows the achieved retransmission rate and throughput for the SD ↔ NC path. Compared to BBR, SATPIPE significantly reduces the retransmission rate from 1.27% to 0.956%. Interestingly, the decrease in retransmission ratio

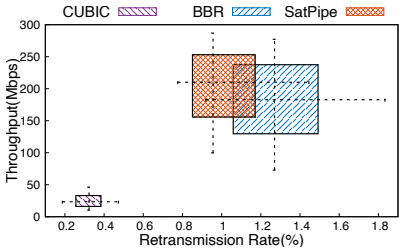


Figure 16. Throughput and retransmission ratio over NC → SD link.

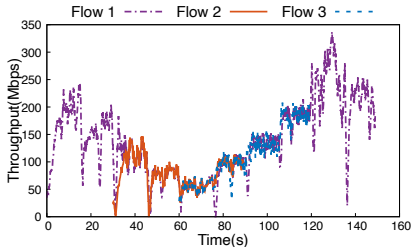


Figure 17. TCP stream fairness verification for multiple data flows

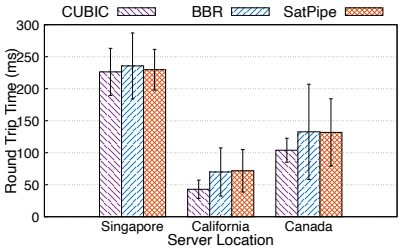


Figure 18. Comparing SATPIPE RTT with existing congestion control algorithms.

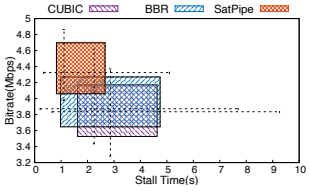
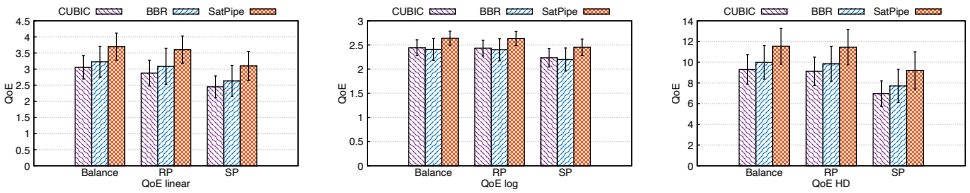


Figure 19. Achieved bitrate and rebuffering time with enough background traffic



(a) QoE linear (b) QoE log (c) QoE HD

Figure 20. Performance under different QoE definitions.

corresponds to the percentage of the handover duration (50ms) relative to the entire handover cycle (15s). It is a solid proof that SATPIPE successfully reduces retransmission during handovers and thus improves bandwidth utilization.

SATPIPE fairness. To understand the impact of competing flows and verify that SATPIPE remains fair across different data streams, we consider a situation where multiple SATPIPE flows share the same Starlink bottleneck link. The experiment is set up such that new SATPIPE flows initiate every 30 seconds and last for 60 seconds. Fig. 17 shows the throughput of three SATPIPE data flows over time. During the initial 30 seconds, only a single SATPIPE flow is active. We observe that SATPIPE reaches good fairness across competing SATPIPE flows when we create or terminate SATPIPE data streams. Table II shows the average throughput for different data flows and verifies that coexisting data streams can all share available bandwidth fairly. Since SATPIPE does not change the logic of BBR for calculating CWND in non-handover situations, SATPIPE and BBR theoretically have the same fairness. During the handover, for all the data streams, SATPIPE will reduce the CWND simultaneously and empty the queue at the same time, which will not affect the fairness among different data streams.

Table II
THROUGHPUT OF MULTIPLE DATA FLOWS

Time Interval	SATPIPE Flow 1 Avg. Throughput	SATPIPE Flow 2 Avg. Throughput	SATPIPE Flow 3 Avg. Throughput
[0,30]s	156.4Mbps	×	×
[35,60]s	89.3Mbps	90.8Mbps	×
[65,90]s	74.2Mbps	77.8Mbps	76.7Mbps
[95,120]s	160.4Mbps	×	159.6Mbps
[125,150]s	225.9Mbps	×	×

SATPIPE latency. To measure TCP RTT, we utilize TCPdump to timestamp packet transmission and receipt of corresponding ACKs. Fig. 18 illustrates the RTT for three representative server locations. SATPIPE shows similar RTT as BBR. We also notice that CUBIC has a significantly lower RTT than BBR and SATPIPE, especially for the SD ↔ NC path. The reason is that CUBIC overly reacts to handover in Starlink, resulting in low bandwidth utilization, and hence low queuing delay.

DASH video streaming QoE. We conduct video streaming measurements to examine how SATPIPE benefits the application layer. The achieved average bitrate and rebuffering time is shown in Fig. 19. SATPIPE achieves 10.8% improvement in bitrate and reduces 33.5% of the rebuffering time, leading to higher QoE. Fig. 20 demonstrates that SATPIPE outperforms state-of-the-art TCP algorithms across all variants of the general QoE definition. Compared to BBR, SATPIPE enhances QoE by 9.6% to 19.4% across these variants. Notably, SATPIPE achieves the highest average improvements of 16.2% in smoothness-favored mode. The primary enhancement provided by SATPIPE in video streaming lies in increasing the playback bitrate and maintaining consistent bitrate before and after handovers, thereby improving smoothness.

VI. CONCLUSION AND FUTURE WORK

We have introduced SATPIPE, an enhanced TCP for highly dynamic satnets that avoids unnecessary throughput fallback caused by handovers. Unlike prior solutions, our algorithm does not rely on satellite tracking and handover reports. It only requires minor modifications to state-of-the-art congestion control algorithms. Our implementation and experiments show that SATPIPE can significantly improve TCP throughput and video QoE. We believe SATPIPE can be adapted to different LEO satellite networks which inevitably experience periodic handover. Our future work will focus on developing more precise physical-layer informed bandwidth estimation algorithms [11], [20] for satnets. The source code of SATPIPE is available at https://github.com/dzhao99/SatPipe

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers for their valuable comments and suggestions that helped improve this paper. This research was supported by Cisco Research.

REFERENCES

[1] SpaceX, "Starlink," 2023, https://www.starlink.com.

[2] Amazon, "Project kuiper," mar 2022, https://www.aboutamazon.com/news/tag/project-kuiper.

[3] OneWeb, "Oneweb," mar 2022, https://oneweb.net/.

[4] Wikipedia, "Starlink," 2024, https://en.wikipedia.org/wiki/Starlink#cite_note-sn2674441-5.

[5] 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.

[6] M. M. Kassem, A. Raman, D. Perino, and N. Sastry, "A browser-side view of starlink connectivity," in *Proceedings of the 22nd ACM Internet Measurement Conference*, 2022, pp. 151–158.

[7] 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.

[8] 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 on Web Conference 2024*, 2024, pp. 2723–2734.

[9] 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.

[10] 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.

[11] X. Xie, X. Zhang, and S. Zhu, "Accelerating mobile web loading using cellular link information," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, 2017, pp. 427–439.

[12] X. Cao and X. Zhang, "Satcp: Link-layer informed tcp adaptation for highly dynamic leo satellite networks," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE, 2023, pp. 1–10.

[13] SpaceX, "Starlink Services LLC ETC Designation Application," https://www.fcc.gov/ecfs/file/download/Starlink%20Services%20LLC%20Application%20for%20ETC%20Designation.pdf, 2023.

[14] T. E. Humphreys, P. A. Iannucci, Z. M. Komodromos, and A. M. Graff, "Signal structure of the starlink ku-band downlink," *IEEE Transactions on Aerospace and Electronic Systems*, 2023.

[15] Z. M. Komodromos, W. Qin, and T. E. Humphreys, "Signal simulator for starlink ku-band downlink," in *Proceedings of the 36th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2023)*, 2023, pp. 2798–2812.

[16] M. Neinavaie and Z. M. Kassas, "Unveiling beamforming strategies of starlink leo satellites," in *Proceedings of the 35th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2022)*, 2022, pp. 2525–2531.

[17] S. Kozhaya, H. Kanj, and Z. M. Kassas, "Multi-constellation blind beacon estimation, doppler tracking, and opportunistic positioning with oneweb, starlink, iridium next, and orbcomm leo satellites," in *2023 IEEE/ION Position, Location and Navigation Symposium (PLANS)*. IEEE, 2023, pp. 1184–1195.

[18] N. Jardak and R. Adam, "Practical use of starlink downlink tones for positioning," *Sensors*, vol. 23, no. 6, p. 3234, 2023.

[19] J. Garcia, S. Sundberg, G. Caso, and A. Brunstrom, "Multi-timescale evaluation of starlink throughput," in *Proceedings of the 1st ACM Workshop on LEO Networking and Communication*, 2023, pp. 31–36.

[20] X. Xie, X. Zhang, S. Kumar, and L. E. Li, "pistream: Physical layer informed adaptive video streaming over lte," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, 2015, pp. 413–425.

[21] FCC. (2023) Spacex non-geostationary satellite system, attachment a, technical information to supplement schedule. https://fcc.report/IBFS/SAT-MOD-20181108-00083/1569860.pdf.

[22] Satmaximum, "Fta universal ku band lnb," 2023, https://satmaximum.com.

[23] Ettus Research LLC, "Universal Software Radio Peripheral (USRP)," n.d., http://www.ettus.com/.

[24] TCPdump, https://satmaximum.com.

[25] Ntppool, https://www.ntppool.org/en/.

[26] Akamai, "dash.js," https://github.com/Dash-Industry-Forum/dash.js/.

[27] T. Roosendaal, "Big buck bunny," in *ACM SIGGRAPH ASIA 2008 computer animation festival*, 2008, pp. 62–62.

[28] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the conference of the ACM special interest group on data communication*, 2017, pp. 197–210.

[29] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over http," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015, pp. 325–338.

[30] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "Bola: Near-optimal bitrate adaptation for online videos," *IEEE/ACM transactions on networking*, vol. 28, no. 4, pp. 1698–1711, 2020.