Low-Latency Live Video Streaming over a Low-Earth-Orbit Satellite Network with DASH

Jinwei Zhao
University of Victoria
Victoria, BC, Canada
clarkzjw@uvic.ca

Jianping Pan
University of Victoria
Victoria, BC, Canada
pan@uvic.ca

ABSTRACT

In light of Starlink's recent rapid growth in constructing a global low-Earth-orbit satellite constellation and offering high-speed, low-latency Internet services, the implications of utilizing Starlink for low-latency live video streaming, particularly in the context of its fluctuating latency and regular satellite handovers events, remain insufficiently explored. In this paper, we conducted a thorough measurement study on the Starlink access network, examining its performance across different protocol layers and at multiple geographical installations, including locations where laser inter-satellite links are utilized in practice. We performed a comprehensive latency target-based analysis of low-latency live video streaming with three state-of-the-art adaptive bitrate (ABR) algorithms in dash.js over Starlink. We presented a novel ABR algorithm designed for low-latency live video streaming over Starlink networks which leverages satellite handover patterns observed from measurements to dynamically adjust video bitrate and playback speed. The performance evaluation of the proposed algorithm was conducted using both a purpose-built network emulator and actual Starlink networks. The results demonstrate that the proposed algorithm effectively delivers a better quality of experience for low-latency live video streaming over Starlink networks, characterized by low live latency, high average bitrate, minimal rebuffering events and reduced visual quality fluctuation.

CCS CONCEPTS

• **Information systems** → **Multimedia streaming**; • **Networks** → **Network performance evaluation**.

KEYWORDS

DASH, Starlink, Low-Latency Live Streaming, ABR

ACM Reference Format:
Jinwei Zhao and Jianping Pan. 2024. Low-Latency Live Video Streaming over a Low-Earth-Orbit Satellite Network with DASH. In *ACM Multimedia Systems Conference 2024 (MMSys '24), April 15–18, 2024, Bari, Italy*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3625468.3647616

MMSys '24, April 15–18, 2024, Bari, Italy

ACM ISBN 979-8-4007-0412-3/24/04…$15.00
https://doi.org/10.1145/3625468.3647616

1 INTRODUCTION

By integrating the distinct capabilities of space, aerial, and terrestrial networks, Space-Air-Ground Integrated Networks (SAGINs) are expected to revolutionize future Internet connectivity through enhanced flexibility and expansible network coverage. In addition to supporting the growing traffic of terrestrial networks, SAGINs will broaden Internet access to remote regions, including rural areas, oceans, and mountainous terrains [12]. Starlink, a division of SpaceX [20], stands out as a pivotal player in offering Internet service through a constellation of low-Earth-orbit (LEO) satellites. These mass-produced small satellites bridge the communication between Starlink user terminals (UTs) and ground stations (GSs). While the concept of satellite Internet is not novel, Starlink's strategy in building a large LEO satellite constellation narrows the bandwidth and latency gap with conventional terrestrial networks. Specifically, SpaceX deploys Starlink satellites at an approximate altitude of 550 km, in contrast to traditional satellite communication networks that rely on either geosynchronous equatorial orbit (GEO) or medium-Earth-orbit (MEO) satellites, which are positioned at higher altitudes and have wider coverage but suffers from higher latency and limited capacity. As of December 2023, SpaceX has more than 5,200 active LEO satellites in operation and attained global coverage in over 70 countries. Nevertheless, SpaceX's constellation ambition extends to launching up to 42,000 LEO satellites and eventually constructing multiple orbital shells [19].

Given the characteristics of LEO satellite networks, UTs utilize phased array antennas to track the moving satellites and perform regular handovers between satellites to maintain network connectivity. Tanveer et al. [24] observed that Starlink employs a global controller for managing the satellite-to-ground scheduling. Specifically, Starlink satellite handover events happen every 15 seconds, at the 12th, 27th, 42nd, and 57th (12-27-42-57) second past every minute, synchronized globally. We conducted an extensive measurement study on the Starlink satellite access network, gateway, point-of-presence (PoP) architectures and global backbone topology [18]. It revealed that the round-trip-time (RTT) from the UT to the GS gateways experiences significant fluctuations and is higher than that of conventional terrestrial Internet access via fiber optics, digital subscriber line (DSL), or cable modem.

Regarding the fluctuating latency in Starlink networks, existing research indicates that Starlink can support a wide range of multimedia services with high-quality assurance, including video-on-demand (VoD) and live streaming, given adequate playback buffers are configured properly. However, the performance remains insufficient for more demanding applications including video conferencing, immersive AR/VR/XR applications, 360-degree and volumetric video streaming and low-latency live (LLL) video streaming.

For VoD services over Starlink, the end-to-end performance is on par with conventional terrestrial networks [27]. Despite frequent handover events and occasional outages of Starlink, the large playback buffer employed in VoD players is usually adequate to compensate for these interruptions, thereby ensuring a smooth viewing experience for end-users. However, in various application scenarios, such as live sports events, cloud gaming, and interactive live broadcasting, the necessity to meet distinct latency targets introduces additional challenges for service providers endeavoring to ensure a low-latency experience, particularly within Starlink's dynamic network environment. Given the widespread adoption of Starlink globally, the demand for LLL video streaming over satellite networks is undeniably significant. O'Hanlon et al. [17] conducted a comprehensive analysis of the performance of three low-latency ABR algorithms in dash.js, namely *Dynamic*, *L2A-LL*, and *LoL+*, considering a variety of latency targets (3, 5.5, 8, and 15 seconds) and configuration options. They employed trace-driven emulations with four different network profiles captured from real-world terrestrial networks. Nonetheless, it remains unknown how the fluctuating latency and frequent satellite handover events affect the performance of LLL video streaming ABR algorithms.

In this paper, we conducted a thorough measurement of the Starlink access network across different protocol layers and geographical Starlink installations, including one where laser inter-satellite links (ISLs) are utilized in practice. We performed a latency target-based analysis of LLL video streaming over Starlink networks. We proposed a novel ABR algorithm based on contextual multi-armed bandit (CMAB) algorithms, specifically designed for live streaming over Starlink networks. CMAB algorithms are lightweight and efficient approaches to online decision-making problems such as the bitrate adaptation problem in live video streaming. They require less computational resources than other reinforcement learning-based algorithms but also provide a competitive performance guarantee with a low regret bound. The proposed CMAB-based ABR algorithm leverages the satellite handover patterns observed from measurements to dynamically adjust the video bitrate and playback speed, thereby ensuring a seamless viewing experience with high average video playback quality, minimal rebuffering events and reduced visual quality fluctuation. The main contributions of this paper are therefore four-fold and can be summarized as follows:

- Assessed the latency and throughput of Starlink access networks across different protocol layers and geographical locations, taking into account scenarios both with and without the utilization of ISLs.
- Conducted a latency target-based measurement and analysis of three state-of-the-art LLL video streaming ABR algorithms in dash.js over Starlink networks.
- Proposed a novel ABR algorithm with satellite handover awareness, improving the Quality of Experience (QoE) of LLL video streaming over Starlink networks.
- Implemented a prototype of the proposed algorithm with dash.js and evaluated its performance using both a purpose-built network emulator and actual Starlink networks.

The remainder of the paper is organized as follows. Section 2 introduces related works in LLL video streaming ABR algorithms and Starlink measurement studies. Section 3 details the testbed setup and outlines the results of our measurements. Section 4 formulates LLL video streaming with CMAB algorithms and presents the design of our proposed algorithm. Section 5 evaluates the performance of the proposed algorithm in both network emulation settings and real Starlink networks. Section 6 discusses the open research challenges and finally, Section 7 concludes the paper, highlighting potential improvement and future work.

2 RELATED WORKS

Over the past decade, numerous ABR algorithms have been proposed to improve the QoE of video streaming in DASH. In this section, our concentration is on three LLL video streaming ABR algorithms available in the dash.js [8] reference player, namely *Dynamic*, *L2A-LL*, and *LoL+*.

The default *Dynamic* [21] algorithm is a hybrid ABR algorithm consisting of throughput-based rule and buffer-based BOLA algorithm [22]. Karagkioules et al. [11] proposed *L2A-LL*, which uses online convex optimization to provide robust video bitrate adaptation strategies without relying on specific parameter tuning, channel model assumptions, throughput estimation or application specific adjustments. Bentaleb et al. [4] introduced *LoL+*, a learning-based ABR algorithm that employs a self-organizing map (SOM) to adapt the bitrate at every segment download boundary. *LoL+* contains four different modules, namely a playback speed control module that combines the current latency and buffer level to control the playback speed; a throughput measurement module that accurately provides throughput estimation based on CMAF chunks; a QoE evaluation module that computes the QoE considering five metrics: selected bitrate, number of bitrate switches, rebuffering duration, latency and playback speed; and a weight selection module that implements a dynamic weight assignment algorithm for the SOM model features.

O'Hanlon et al. [17] conducted a latency target-based analysis of three ABR algorithms concerning a range of latency targets (3, 5.5, 8, and 15 seconds) and configuration options for the LLL video streaming performance. The *Dynamic* algorithm performs the best in terms of low rebuffering duration, with the least number of stalls and the shortest overall rebuffering time. In terms of live latency, the *Dynamic* algorithm also provides the smallest deviation from the latency target in all the scenarios evaluated and provides the most stable but lower video bitrate quality, whilst *L2A-LL* and *LoL+* can reach a higher video quality level. They further demonstrated the impact of the `FastSwitching` option in dash.js. When enabled, this option replaces low-bitrate video segments in the playback buffer with high-bitrate segments during a quality increase phase, rather than appending them directly to the end of the current playback buffer. However, the `FastSwitching` option brings a significant number of re-requests that consume more bandwidth but do not generally increase the QoE. Thus, the option is recommended to be disabled in LLL video streaming.

Since the launch of Starlink's beta testing in 2020, it has attracted considerable research interest from both the industry and academia. Research topics span from network measurements [18, 24, 27] to physical layer signal structure analysis [5] and routing protocol design [26], among others. Zhao et al. [27] conducted

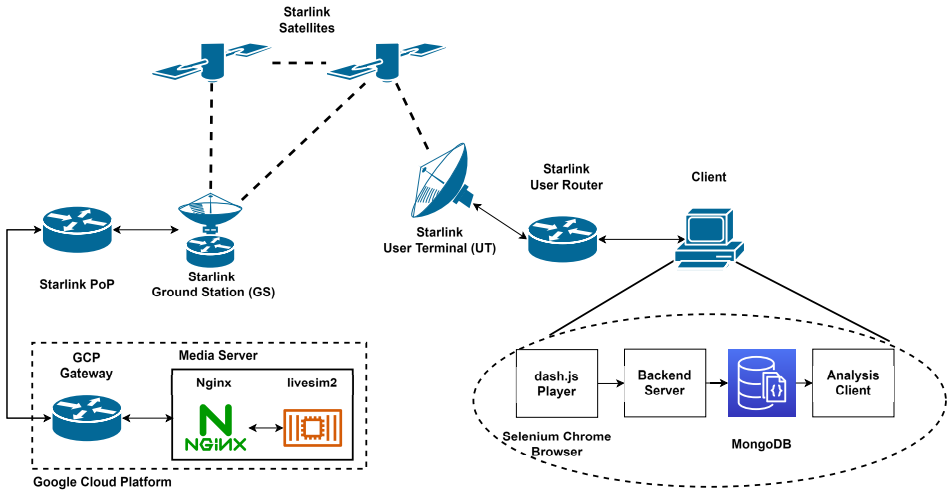


Figure 1: Starlink DASH testbed architecture

a systematic measurement on real-time multimedia services over Starlink, including VoD (YouTube), live video streaming (Twitch) and video conferencing service (Zoom). The Starlink network typically delivers satisfactory performance for multimedia services. However, factors like extreme weather, satellite handover events, and changes in packet routing paths can impact its performance. VoD services remain largely unaffected due to their substantial playback buffers, which can compensate for Starlink's occasional short-time outages and frequent satellite handover events. In contrast, interactive applications such as video conferencing and live video streaming, face more pronounced performance challenges. Tanveer et al. [24] noted a consistent pattern in Starlink's satellite handover events occurring every 15 seconds. Specifically, these events manifest in the latency characteristics at the 12th, 27th, 42nd, and 57th seconds of each minute. We provided comprehensive insights into the access network, gateway, PoP, and backbone network architectures of Starlink, illustrating the findings with detailed network topology diagrams, derived from collaborative measurements across the world [18].

3 MEASUREMENT

In this section, we first present our testbed setup and the measurement results of Starlink networks. We then outline the approach behind our latency target-based analysis of LLL video streaming over Starlink networks. The subsequent measurement results for LLL video streaming, in conjunction with the performance evaluation of the proposed algorithm, are presented in Section 5.

3.1 Setup

The architecture of our Starlink measurement testbed is shown in Figure 1. For each Starlink installation, we deployed a virtual machine in the nearest Google Cloud Platform (GCP) availability zone as the end-to-end latency measurement target and the media server for LLL video streaming. For example, the Starlink installation in the Pacific Northwest is associated with the Starlink Seattle PoP. To minimize additional terrestrial network latency, we deployed the media server in GCP's *us-west1-a* availability zone, physically located in Oregon, USA. Our measurements indicate that the network latency between the Starlink Seattle PoP and the GCP gateway in this availability zone is below 10 milliseconds, which is negligible compared to the latency of Starlink access networks. We also have access to a Starlink installation in Seychelles, located in the western Indian Ocean, proximate to the eastern coast of Africa. As of December 2023, the only Starlink PoP in Africa is physically located in Lagos, Nigeria [15], which is on the western coast of the continent. Starlink releases a GeoIP feed detailing their customer IP allocations[1], along with DNS PTR records that indicate the associated home PoP locations[2]. In Africa, Starlink primarily serves its subscribers through the Lagos PoP, while occasionally re-associating users to the Frankfurt PoP and London PoP, possibly due to load balancing and capacity considerations. Throughout our measurement, the Starlink installation in Seychelles was associated with the Lagos PoP. Considering the absence of Starlink GSs within a 5,000 km radius of this Starlink installation, our inference is that the packets traverse multiple laser inter-satellite links, commonly referred to as ISLs, before being relayed to the GS, reaching the Lagos PoP, and subsequently access the Internet. We deployed the media server for this region in GCP's *europe-southwest1-a* availability zone, physically located in Madrid, Spain, where the Starlink Madrid PoP is interconnected with Starlink Lagos PoP through Starlink's terrestrial backbone infrastructure [18].

We deployed network measurement scripts and the LLL video streaming stack on a mini PC or virtual machine directly connected to the Starlink user router via Ethernet. The LLL video streaming stack consists of a modified dash.js player, a backend server, a MongoDB database and an analysis client. The modified dash.js player sends the playback metrics to the backend server through REST APIs, which are then stored in the MongoDB database. The analysis client queries the MongoDB database after playback sessions

[1] https://geoip.starlinkisp.net/feed.csv
[2] https://starlink-enterprise-guide.readme.io/docs/peering-with-starlink

Table 1: Starlink access network latency (ms) to GS gateway

Starlink installations	Min	Median	Average	σ
Without ISL	16.7	42.8	47.5	20.9
With ISL	59.1	109.0	119.8	43.7

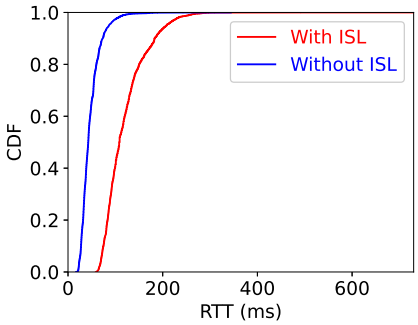
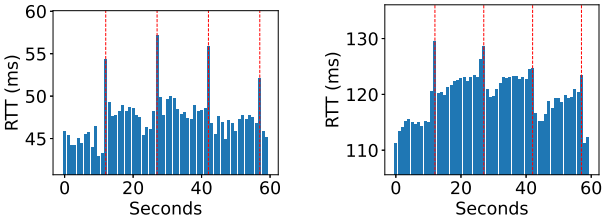


Figure 2: CDF of the RTT to GS gateway



(a) Average RTT without ISL **(b) Average RTT with ISL**

Figure 3: Average RTT to gateway at every second

and generates corresponding figures. On each media server, we deployed `livesim2` [9], a DASH live source simulator implemented in Go by the DASH Industry Forum, which takes segmented DASH source videos as the input assets, and produces a wall-clock (UTC) synchronized linear stream of video segments. By looping the input VoD DASH assets and dynamically adjusting the timestamps, a perpetual "live" video stream is made available to the clients. Nginx is installed as the frontend web server to serve the MPD files and the corresponding "live" video streams.

3.2 Measuring Starlink Access Network

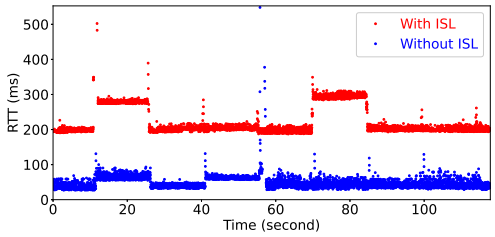
For regular Starlink subscribers, the Starlink user dish can always reach the GS gateway in one IP hop, at a carrier-grade NAT (CGNAT) address `100.64.0.1` [18]. For those on the business or priority subscription plan with the public address option, they are allocated a public IPv4 address bounded to their Starlink user routers, which is reachable from the Internet.

Our first measurement is to evaluate the latency performance of Starlink access networks. We started with measuring the RTT between clients and the GS gateway using ICMP `ping`. The interval for the `ping` command is set at 10 milliseconds. We continuously conducted our systematic latency measurements since November 2023 and released the dataset to the public [28]. In this paper, we show a snapshot of the measurement results from the Starlink installations in the Pacific Northwest and Seychelles, with a total continuous duration of 60 minutes. A summary of the Starlink access network latency to the GS gateway is shown in Table 1.

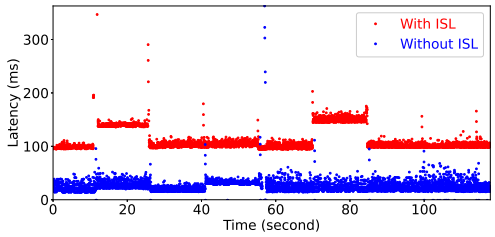
For the Starlink installation without ISL, the overall access latency can easily be maintained at around 50 milliseconds, with the minimal RTT being 16.7 milliseconds, the median RTT being 42.8 milliseconds and the average RTT being 47.5 milliseconds. For the Starlink installation with ISL, the overall access latency fluctuated more than the one without ISL, with the minimal RTT being 59.1 milliseconds, the median RTT being 109.0 milliseconds, the average RTT being 119.8 milliseconds and a higher standard deviation σ=43.7 milliseconds than σ=20.9 milliseconds without ISL. The cumulative distribution function (CDF) of both latency distributions is shown in Figure 2. We calculated the average RTT to the GS gateway at every second for both Starlink installations, as shown in Figure 3. It revealed an obvious pattern that at (12-27-42-57) seconds of every minute, the average RTT between Starlink UTs and their corresponding GS gateways spikes, which indicates that satellite handover events happen at synchronized seconds globally at different geographical locations. This observation is consistent with the findings in [24]. However, our observations in Figure 3 indicate that the potential impact of satellite handover events is more relatively pronounced on the Starlink installation that did not utilize ISL during our measurement.

To gain deeper insights into how satellite handover events influence the end-to-end (E2E) performance of different applications, we carried out time-synchronized measurements of latency and TCP throughput, as indicated in Figure 4. E2E latency and throughput are measured with `IRTT` and `iPerf3` respectively. We deployed `IRTT` and `iPerf3` daemon programs on the media servers as illustrated in Section 3.1 and Figure 1. All the media servers and clients are configured with NTP time synchronization using `chrony` and Google Public NTP service[3], such that `IRTT` can provide accurate one-way delay (OWD) measurements. `IRTT` provides latency measurements by sending UDP packets on a fixed time interval regardless of whether replies are received and it is not affected by the Linux kernel scheduling or other factors that can affect the packet interval of ICMP ping measurements. By utilizing UDP instead of ICMP, it can also avoid the potential ICMP deprioritization on some network devices and provide more realistic measurement results close to real-world applications. In our measurements, the `IRTT` request interval is set to 10 milliseconds, consistent with the ICMP `ping` experiments. To measure the throughput, we utilized `iPerf3` and set the report interval to 100 milliseconds, which is the minimal `iPerf3` report interval. In this paper, our primary focus was on evaluating the downlink throughput performance from the media server to the video streaming clients, aligning with typical video streaming scenarios. However, a brief discussion on the uplink performance for live broadcasting scenarios is provided in Section 6. A single TCP flow was used for all the `iPerf3` throughput measurements and the TCP congestion control algorithm used

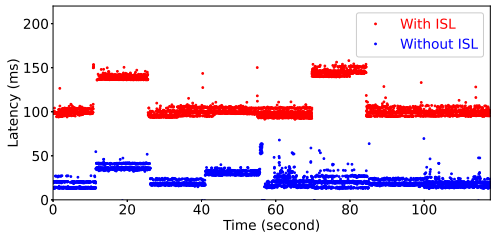
[3]https://developers.google.com/time



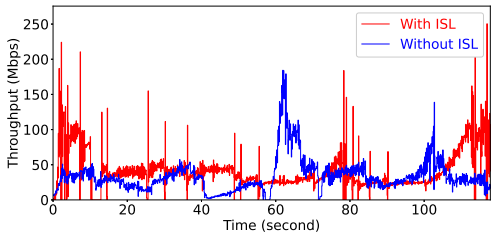
(a) E2E latency



(b) Uplink OWD



(c) Downlink OWD



(d) Downlink throughput

Figure 4: Time synchronized latency and throughput measurements

is CUBIC, which is the default congestion control algorithm in the Ubuntu 22.04 operating system we used.

Figure 4 shows a 2-minute window of the time-synchronized latency and throughput measurements. There is a notable variance in latency patterns across each 15-second interval. At the boundaries of each timeslot, both the uplink and downlink OWD exhibit surges, aligning with the satellite handover events. Regarding OWD, the downlink OWD exhibits a more pronounced "strip band" pattern compared to the uplink OWD, especially for Starlink installations without ISL in Figure 4(c). This distinction likely arises because downlink access is allocation-based, designating specific timeslots in a media access frame for a particular UT. Conversely, the uplink operates on either a contention-based system or a poll-randomize-grant mechanism [10]. Figure 4(d) shows the instantaneous downlink throughput measurements. It shows that while the RTT might stay relatively stable across different timeslots as the satellite handover events happen, the client and server might have to go through the TCP slow start pattern because of RTT timeout or due to packet loss, which can significantly impact the TCP throughput performance. It is important to note that the utilization of ISL does not directly impact downlink throughput. Instead, it is influenced by Starlink's capacity limitations and Quality of Service (QoS) policies in different regions.

Table 2: Bitrate ladder of the LLL video dataset

Resolution	Frame rate (fps)	Encoding bitrate (Kbps)
1920x1080	50	6000
1920x1080	25	5100
1920x1080	50	4900
1024x576	25	1500
1024x576	25	1200
768x432	25	900
512x288	25	450
480x270	12.5	300

3.3 Measuring LLL Video Streaming

Our measurement on LLL video streaming over Starlink networks is based on dash.js v4.7.1, which was released in June 2023. Three ABR algorithms, namely *Dynamic*, *LoL+* and *L2A-LL* and four different latency targets were evaluated. The latency targets range from 3 seconds to 6 seconds. The video dataset used in our measurements is obtained from the CTA WAVE Test Project[4]. The bitrate ladder is shown in Table 2, which contains 8 representations with different resolutions and frame rates, and the target H.264 encoding bitrate ranges from 300 Kbps to 6000 Kbps. The video is segmented into 2-second aligned video segments. We played back the "live" video stream produced by `livesim2` for 5 minutes in each round of measurement and repeated the same measurement 10 times for each latency target and ABR algorithm. The following metrics are collected every 100 milliseconds on our modified dash.js player during playback,

- Average live latency
- Average bitrate
- Rebuffering time ratio (%)
- Number of bitrate switches
- Bitrate standard deviation

and they are sent to the backend server and stored in the MongoDB databases for the analysis client to evaluate the performance of LLL video streaming ABR algorithms. The measurement results for LLL video streaming over Starlink networks, along with a detailed performance evaluation and comparison with the proposed algorithm are shown in Figure 6 to Figure 10.

[4]https://github.com/cta-wave/Test-Content

Table 3: Summary of key notations

Notation	Definition
K	Number of arms
T	Total number of rounds
$a(t)$	Arm selected by agent at round t
$b(t)$	Context vector revealed to agent at round t
μ_k	Distribution parameter for arm k
q	First round of the current 15-second timeslot
$\mathcal{H}_q^{t-1}$	History beginning from q up to round $t-1$
$a^*(t)$	Optimal arm at round t
$r(i)$	Reward for video segment i
C	Total number of rebuffering events
C_k	Total number of rebuffering events at bitrate k
t_j	Rebuffering time for event j
t_j^k	Rebuffering time for event j at bitrate k
$\mathrm{B}_{\max}$	Highest video bitrate available in the MPD file
$\mathrm{R}(t)$	Playback speed at round t
$\mathrm{X}(t)$	Estimated network throughput at round t
$\mathrm{L_N}(t)$	Measured network latency at t
$\mathrm{Buffer}_{\min}$	Minimal playback buffer threshold
$\mathrm{Buffer}_{\mathrm{current}}$	Current playback buffer level
$\mathrm{LL}_{\mathrm{target}}$	Latency target for the playback session
$\mathrm{LL}_{\mathrm{current}}(i)$	Playback latency when downloading segment i
$\mathrm{QoE}_{\mathrm{P1203}}(i)$	ITU-T P.1203 QoE score for segment i
$\mathrm{QoE}(i)$	Final QoE for segment i

4 PROBLEM FORMULATION

In this section, we present the problem formulation of LLL video streaming using CMAB algorithms, along with a novel QoE-driven reward function and catch-up policies to improve the QoE of LLL video streaming over Starlink networks. The key notations are summarized in Table 3.

4.1 System Model

We follow the general CMAB algorithm setting as in [3] to model the LLL video streaming scenario as follows. The video bitrate adaptation problem in live video streaming can be formulated as an online decision-making process. The video player, referred to as an agent, is presented with a bitrate ladder of K different bitrate levels to choose from in each of T rounds. We assume T is finite but unknown to the agent. The K video bitrate levels are referred to as K arms. The decision on which bitrate should be selected for playback is analogous to choosing an arm to pull by the agent. Before the agent pulls an arm in each round, a context vector $b(t) \in \mathbb{R}^d$ is presented, which contains the context information when the current round happens. The context information can include metrics such as the current network latency, current video playback speed, estimated network throughput, etc. In this paper, we define the context vector $b(t)$ as follows,

$$b(t) = [\mathrm{L_N}(t), \mathrm{R}(t), \mathrm{X}(t)] \tag{1}$$

where at round t, $\mathrm{L_N}(t)$ is the latest measured network latency to the media server, $\mathrm{R}(t)$ is the current playback speed, and $\mathrm{X}(t)$ is the estimated network throughput, respectively.

The agent chooses an arm that is anticipated to yield the highest expected reward in the current round. That is, the video bitrate selection should yield the highest expected QoE in the current round, without causing playback interruptions or rebuffering events. In each round, only the reward of the chosen arm is revealed to the agent, leaving the rewards of the unselected arms undisclosed.

A history $\mathcal{H}^{t-1}$ containing all the previous rewards of the selected arms and their respective contexts up to round $t-1$ can be compiled by the agent before round t. In this paper, we only consider the history $\mathcal{H}_q^{t-1}$ during the current 15-second timeslot beginning from round q,

$$\mathcal{H}_q^{t-1} = \{k, r_k(s), b(s), s = q, \ldots, t-1\} \tag{2}$$

where k denotes the arm played at round s and $r_k(s)$ is the reward for arm k at round s, $b(s)$ is the context vector at round s, and q is the first round of the current 15-second timeslot.

Given $b(t)$, the reward for arm k at round t is derived from an unknown distribution with mean $b(t)^T\mu_k$, where $\mu_k \in \mathbb{R}^d$ is a constant parameter unknown to the agent. $b(t)^T$ denotes the matrix transpose of $b(t)$. The expected reward of $r_k(t)$ for each arm k given $b(t)$ and $\mathcal{H}_q^{t-1}$ can be defined as,

$$\mathbb{E}\left[r_k(t) \mid b(t), \mathcal{H}_q^{t-1}\right] = b(t)^T\mu_k. \tag{3}$$

In a CMAB scenario, an agent employing an online learning algorithm must decide which arm k to pull at each round t, considering both the history $\mathcal{H}_q^{t-1}$ and the context vector $b(t)$ of the current round made available to the agent.

Define $a^*(t)$ as the optimal arm at round t that provides the maximum expected reward, formulated as $a^*(t) = \arg\max_k b(t)^T\mu_k$. Let $\Delta_k(t)$ represent the difference in reward between the optimal arm $a^*(t)$ and arm k at time t, i.e.,

$$\Delta_k(t) = b(t)^T\mu_{a^*(t)} - b(t)^T\mu_k \tag{4}$$

Then, the regret at round t is defined as

$$\mathrm{regret}(t) = \Delta_k(t) \tag{5}$$

It is worth noting that the CMAB problem setting presented here deviates from the one outlined in [3]. In our scenario, the assumption is that each arm k is revealed with the identical context vector $b(t)$. Moreover, each arm follows an unknown yet unique distribution defined by its respective μ_k. We also only consider the history $\mathcal{H}_q^{t-1}$ during the current 15-second timeslot beginning from round q, because of the unique and fluctuating latency pattern in each timeslot as observed in Figure 4.

4.2 QoE-driven Reward Function

To evaluate the video streaming experience, we employed ITU-T P.1203 [2] Recommendation model to derive the QoE score for each video segment. Specifically, we used the ITU-T P.1203 Mode 0, which derives from video stream metadata and yields an overall QoE score represented as a Mean Opinion Score (MOS) for video segments. We took the O.46 score from ITU-T P.1203 model outputs, which is a single media session quality score, on a 1–5 quality scale in real numbers.

To integrate the ITU-T P.1203 model with our CMAB-based algorithm, we develop a QoE-driven reward function with the primary objective of reducing the live latency, increasing playback

bitrate and minimizing rebuffering events. The reward function is defined as follows,

$$\mathrm{QoE}(i) = \mathrm{QoE}_{\mathrm{P1203}}(i) * \frac{\mathrm{LL}_{\mathrm{target}}}{\mathrm{LL}_{\mathrm{current}}(i)} * \frac{k}{\mathrm{B}_{\mathrm{max}}} - \frac{\sum_{j=1}^{C_k} t_j^k}{\sum_{j=1}^{C} t_j} \quad (6)$$

where $\mathrm{QoE}_{\mathrm{P1203}}(i)$ represents the O.46 MOS score for video segment i calculated by ITU-T P.1203 model, $\mathrm{LL}_{\mathrm{target}}$ represents the latency target in the current playback session, $\mathrm{LL}_{\mathrm{current}}(i)$ represents the live latency when downloading video segment i, C represents the total number of rebuffering events, C_k represents the number of rebuffering events happens at bitrate k, t_j represents the duration of rebuffering event j, t_j^k represents the duration of rebuffering event j happens at bitrate k, arm k is the video bitrate for segment i, and $\mathrm{B}_{\mathrm{max}}$ is the highest video bitrate available in the MPD file.

The agent pulls an arm before downloading each video segment i, and the corresponding reward $r(i)$ for video segment i obtained is defined as,

$$r(i) = \mathrm{QoE}(i) \quad (7)$$

The objective of the agent is to pull the best arm which yields the highest expected reward in each round for video segment i.

4.3 Catch-up Policy

A similar empirical catch-up policy as in *LoL+* [4] is employed in the proposed algorithm. The main goal of our catch-up policy and playback speed control module is to avoid potential playback interruptions during satellite handover periods, in addition to the case when the playback buffer is below the safe threshold. The satellite handover period is defined based on the measurement results in Section 3, as the (12-27-42-57) seconds of every minute. We modify the *LoL+* catch-up policy as follows,

- The current playback buffer level is below the safe threshold ($\mathrm{Buffer}_{\mathrm{min}}$), or it is currently within the satellite handover period: slow down the playback speed below 1.0.
- The current buffer level is sufficient, and it is not within the satellite handover period:
  - The live latency is close to the latency target ($\epsilon = \pm 2\%$): maintain playback speed at 1.0.
  - The live latency is lower than the latency target: slow down playback speed.
  - The live latency is higher than the latency target: speed up playback speed.

5 EVALUATION

We evaluated the performance of the proposed algorithm and compared it with the other three LLL video streaming ABR algorithms in network emulation settings and real Starlink networks. The performance evaluation on Starlink networks was only conducted on the Starlink installation without ISL being utilized. The implementation of this paper is available on GitHub[5].

In addition to evaluating the performance of LLL video streaming ABR algorithms in real Starlink networks, we also built a network emulation testbed to provide repeatable environments for performance evaluation. The architecture of the purpose-built emulation testbed is similar to the real Starlink testbed with minor modifications, which is shown in Figure 5. All the components in Figure 5 are deployed on a single machine using Docker containers and orchestrated by Docker Compose. A "Traffic Shaper" container is added to the emulation testbed between the dash.js player and the frontend Nginx web server to add artificial network latency and packet loss following the Starlink satellite handover pattern and the measured latency performance. Specifically, the "Traffic Shaper" container is implemented using the Linux network utility `tc` and `netem` to introduce artificial delay and packet loss during handover periods. We utilized latency traces from our `LENS` dataset [28] to provide reasonable and realistic network latency and packet loss estimations during handover periods. In our emulation, we set the latency to 100 ms and the packet loss rate to 1% during handover periods, which is defined as (12-27-42-57) seconds every minute. The network latency and packet loss rate outside the handover periods are set to 40 ms and 0.1% respectively.

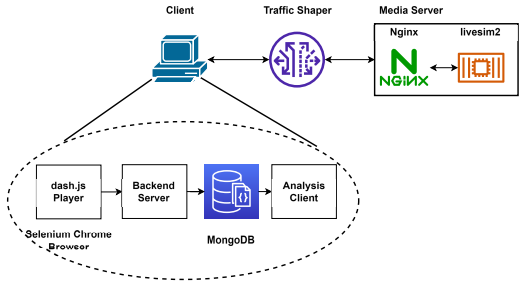
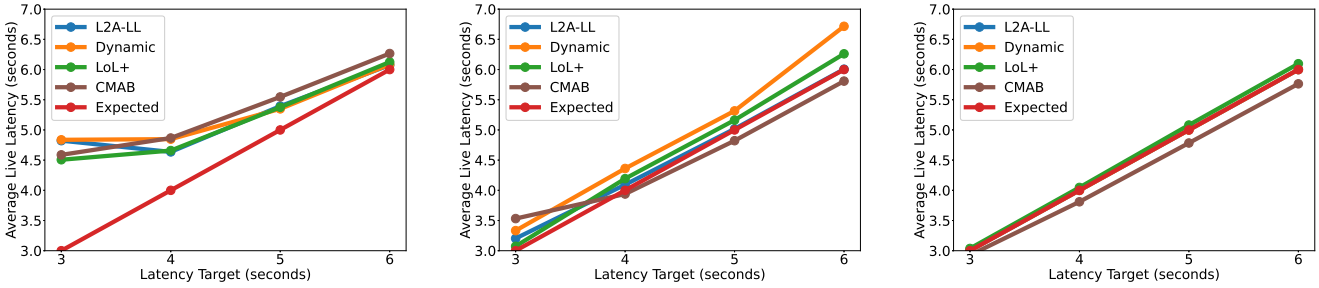


Figure 5: Starlink DASH emulation testbed architecture

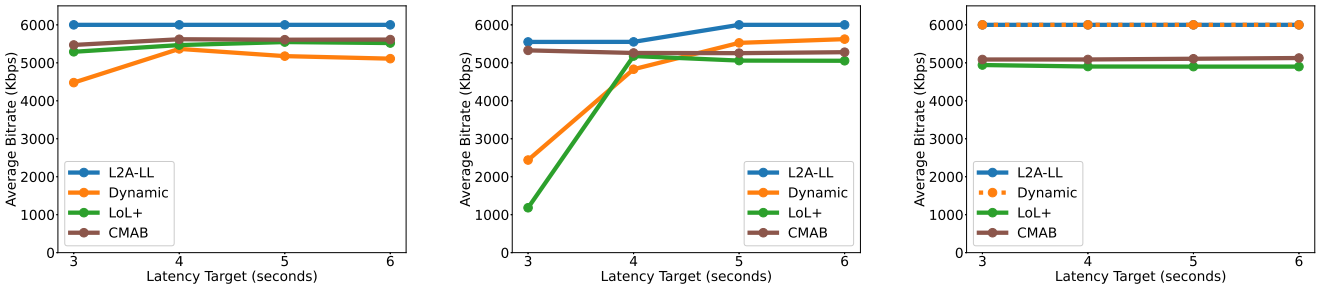
We implemented our CMAB-based ABR algorithm on dash.js v4.7.1 and built an end-to-end evaluation prototype. To solve the online learning problem with CMAB algorithms, `MABWiser` [23] which provides fast prototyping with various CMAB algorithms is chosen in our implementation. While other CMAB algorithms are available, our implementation chose Linear Thompson Sampling (LinTS) [3] as our solution. To integrate `MABWiser` with dash.js, we utilized `Pyodide` [25], which is a WebAssembly-based Python runtime for browsers. We implemented the core CMAB algorithm for bitrate adaptation and QoE calculation in Python and interacted with dash.js through `Pyodide` APIs. We acknowledge that implementing our proposed algorithm in dash.js with WebAssembly-based Python is slower than a native JavaScript implementation, with our measurement, the performance overhead and the time cost to solve the CMAB problem in each round is below 100 milliseconds, which is negligible in our scenario. We also disabled the `FastSwitching` option in dash.js similar to the latency target-based measurements in [17] and used the "moof" parsing method for throughput calculation. The network latency to the media server $\mathrm{L}_{\mathrm{N}}(t)$ is measured in the backend server as shown in Figure 1 and Figure 5 and queried by dash.js clients through REST APIs. We set the *maxDrift* to 5s and *playbackRate* to 0.17. For *Dynamic* and *L2A-LL* algorithms, we set the catch-up mechanism to the *Default*, while *LoL+* and our proposed algorithm have their distinct catch-up mechanisms. The exploration rate of LinTS is set to 1.0. In addition to network emulation and real Starlink networks, we also

[5]https://github.com/clarkzjw/mmsys24-starlink-livestreaming



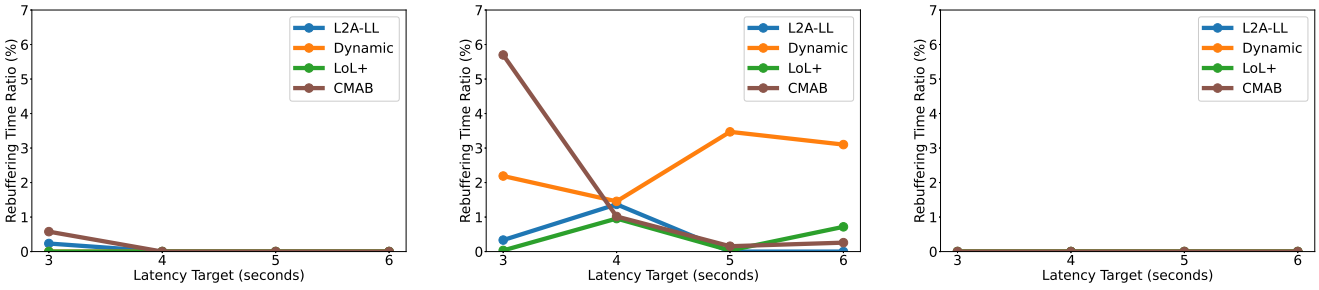
(a) Emulation (b) Starlink (c) Terrestrial

Figure 6: Average live latency



(a) Emulation (b) Starlink (c) Terrestrial

Figure 7: Average bitrate

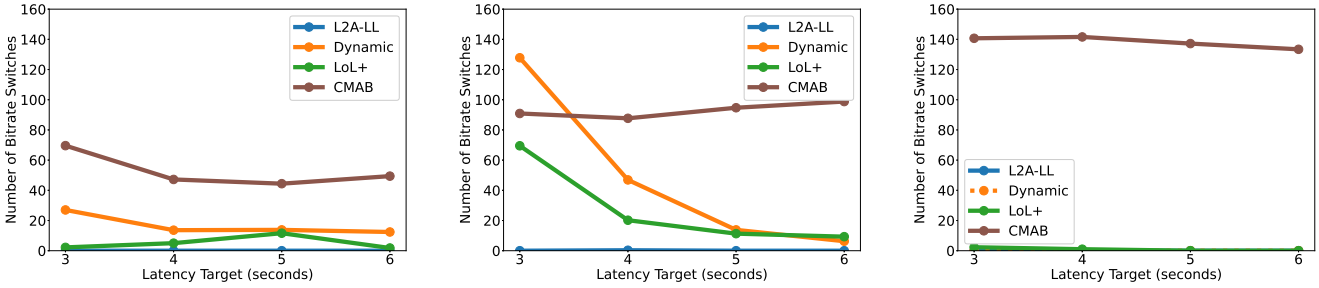


(a) Emulation (b) Starlink (c) Terrestrial

Figure 8: Rebuffering time ratio (%)

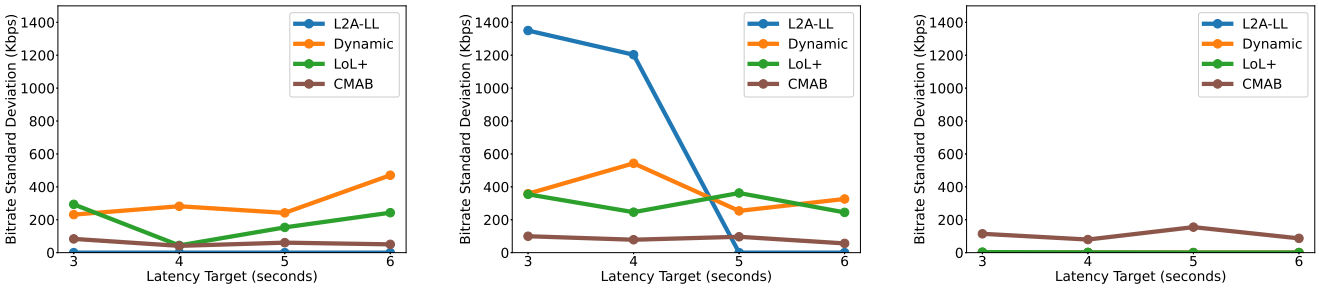
included a terrestrial network setting as the control group. As our proposed ABR algorithm takes advantage of predictable satellite handover patterns, when evaluating the performance of the proposed algorithm in the terrestrial network setting, we disabled the handover awareness and replaced the catch-up mechanism with the *Default* catch-up policy in dash.js.

The first performance metric is average live latency, which plays the most critical role and significantly affects user experience in LLL video streaming. Figure 6 shows the average live latency in all three network settings. In the network emulation scenario as shown in Figure 6(a), when the latency target is 3 seconds, all the ABR algorithms cannot reach the expected latency requirement with at least 1.5 seconds of deviation. As the latency target increases, the gap between average live latency and expected latency requirement narrows. This is partially because of the 2-second segment duration in our performance evaluation, which leaves the catch-up policy with less space to adjust the playback speed to reach the latency target. In real Starlink networks, as shown in Figure 6(b), only the proposed algorithm can achieve lower live latency than the expected latency target, when the latency target is larger than 4 seconds. In the terrestrial network setting as shown



(a) Emulation (b) Starlink (c) Terrestrial

Figure 9: Number of bitrate switches



(a) Emulation (b) Starlink (c) Terrestrial

Figure 10: Bitrate standard deviation

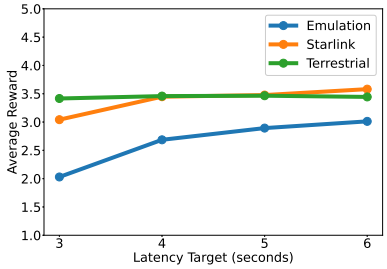


Figure 11: Average reward

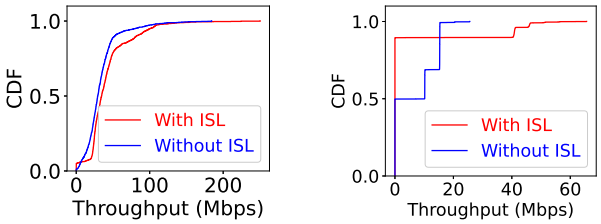
in Figure 6(c), the proposed algorithm has the lowest average live latency across different latency targets.

In Figure 7, we can see all ABR algorithms show the same trend in increasing the average bitrate as the latency target increases. Across different scenarios, *L2A-LL* maintains the highest average bitrate, while *Dynamic* has larger bitrate fluctuations in network emulation and real Starlink networks. The proposed CMAB-based ABR algorithm always maintains a high average bitrate across different latency targets and different network scenarios. Although the proposed algorithm has a relatively lower average bitrate than *Dynamic* and *L2A-LL* in the terrestrial network setting, it is a beneficial trade-off when considering the performance in dynamic network environments.

Rebuffering events interrupt the live video streaming playback and significantly affect user experience. Figure 8 shows the rebuffering time ratio in all three network settings. A larger latency target allows the player to build a more substantial local playback buffer, thereby better tolerating network fluctuations and reducing rebuffering events. In the real Starlink network setting, the proposed algorithm can achieve a low rebuffering time ratio close to *L2A-LL* as shown in Figure 8(b), especially when the latency target is larger than 4 seconds. In Figure 8(a), the deviation of the rebuffering time ratio is less significant than the real Starlink network setting, partially due to the less dynamic network emulation setting than real Starlink networks. No rebuffering events are observed in the terrestrial network setting for all ABR algorithms as shown in Figure 8(c).

Figure 9 shows the number of bitrate switches in all three network settings. *L2A-LL* has the least number of bitrate switches across different latency targets. Both *Dynamic* and *LoL+* show a similar decreasing trend in the number of bitrate switches as the latency target increases in network emulation and real Starlink networks as shown in Figure 9(a) and Figure 9(b). Our proposed algorithm has a generally higher number of bitrate switches than other ABR algorithms in all network scenarios.

However, Figure 10 shows that our proposed algorithm has a relatively low bitrate standard deviation in both network emulation and real Starlink networks. A lower bitrate standard deviation indicates that the algorithm can maintain a more stable playback



(a) Downlink throughput CDF **(b) Uplink throughput CDF**

Figure 12: CDF of throughputs

quality and the shifts in playback quality with the proposed algorithm bring less visual impact to users compared with other ABR algorithms. However, this implication should be further verified by other visual quality assessment methods such as VMAF [16] in future works. There is one observation from the results worth mentioning is that our proposed algorithm also has a high number of bitrate switches in the terrestrial network environment. The potential explanation is that our algorithm only considers the history within the current 15-second timeslot without a moving average of previous measurements, and it has to restart the exploration phase from scratch after every handover event. In conjunction with the 2-second DASH video segment duration, and the limited playback buffer size in live video streaming, it is challenging for the agent to converge to the optimal bitrate in a short time.

Figure 11 shows the average reward of the proposed algorithm in all three network settings. It shows that our ITU-T P.1203-based reward function can effectively guide the CMAB algorithm to make ABR decisions and it can accurately reflect the performance of the proposed algorithm in different network scenarios.

6 DISCUSSION

In this paper, we mainly focused on measuring the performance of Starlink networks and low-latency ABR algorithms in one-way live video streaming scenarios from media servers to end users. However, there are still many open research challenges left to be explored in the future.

As the popularity of live broadcasting and short video-sharing platforms such as Twitch and TikTok grows, the video ingest performance on the uplink is also becoming more important. However, the uplink performance of Starlink networks is significantly subpar compared to the downlink performance. As shown in Figure 12, the uplink throughput is significantly lower than the downlink throughput with very high fluctuations. Similar to Section 3.2, it is important to note that the utilization of ISL does not directly impact throughput performance. Instead, it is influenced by Starlink's capacity limitations and QoS policies in different geographical regions. In contrast to the "pull"-based model of live video streaming on the downlink path, live broadcasting workflow involves streaming software such as OBS and uses RTMP/HLS/DASH to ingest video streams to the service provider's servers using a "push"-based model. In this case, the video ingestion clients can better utilize the predictable satellite handover patterns to dynamically adjust the video encoding bitrate and sending rate to avoid potential bufferbloat and improve the video ingestion performance.

The dynamic nature of LEO satellite networks poses challenges to the traditional CDN architecture for media delivery. Given the inherent mobility of LEO satellites, the conventional paradigms of "local" or "edge" computing are being redefined, leaving issues such as resource allocation, storage, and caching as open areas for exploration. The TCP performance over Starlink is significantly affected by the slow start pattern as satellite handover events happen every 15 seconds. Other generic congestion control algorithms, such as BBR and HyStart++, or congestion control algorithms with satellite network awareness, such as SaTCP [6], may yield better TCP throughput performance when compared to CUBIC on Starlink networks. QUIC and its extensions including MPQUIC [13] and Media-over-QUIC (moq) [7] can potentially significantly reduce the impact of satellite handover events on TCP performance and therefore improve upper-layer application performance.

The performance of demanding applications such as cloud gaming over LEO satellite networks [14] remains an area requiring comprehensive investigation. Essentially, cloud gaming is interactive ultra-low latency live video streaming. Latency-sensitive scenarios, especially in games like first-person shooters, are particularly vulnerable to fluctuating latency and frequent satellite handovers, given they also encompass human interactions via Starlink's uplink channels.

7 CONCLUSION

In this paper, we conducted a thorough measurement study on the Starlink access network at multiple geographical Starlink installations, across different protocol layers from access latency and raw TCP throughput to application-layer LLL video streaming performances. We proposed a CMAB-based ABR algorithm for LLL video streaming over Starlink networks, with a novel reward function and catch-up policy considering satellite handover patterns. An end-to-end prototype of the proposed algorithm was implemented in dash.js and performance evaluations were conducted on a purpose-built network emulation testbed and real Starlink networks. The results illustrated that the proposed algorithm can achieve low playback latency, high video bitrate, low rebuffering time ratio and few visual quality fluctuations. For future works, VMAF can be used to evaluate the visual quality fluctuation of different ABR algorithms in LLL video streaming over Starlink networks. Utilizing CMAF chunked encoding and chunked transfer over Starlink networks could further enhance the live streaming latency and QoE. It is also worth investigating the performance of existing LLL video streaming ABR algorithms with satellite handover awareness and the performance of uplink video streaming over ISL-enabled Starlink networks.

ACKNOWLEDGMENT

We appreciate the constructive comments and feedback from the reviewers and the shepherd. This work was supported in part by NSERC, CFI and BCKDF. The work is also not possible without Starlink users such as Dominique who allowed us to access their dishes.

REFERENCES

[1] [n. d.]. CTA The Wave Project: Web Application Video Ecosystem Interoperability Project. https://github.com/cta-wave/Test-Content.

[2] 2017. P.1203 : Parametric Bitstream-Based Quality Assessment of Progressive Download and Adaptive Audiovisual Streaming Services over Reliable Transport. https://www.itu.int/rec/T-REC-P.1203-201710-I/en.

[3] Shipra Agrawal and Navin Goyal. 2013. Thompson Sampling for Contextual Bandits with Linear Payoffs. In *Proceedings of the 30th International Conference on Machine Learning*. PMLR, 127–135.

[4] Abdelhak Bentaleb, Mehmet N. Akcay, May Lim, Ali C. Begen, and Roger Zimmermann. 2022. Catching the Moment With LoL$^+$ in Twitch-Like Low-Latency Live Streaming Platforms. *IEEE Transactions on Multimedia* 24 (2022), 2300–2314. https://doi.org/10.1109/TMM.2021.3079288

[5] Rodrigo Blázquez-García, Diego Cristallini, Martin Ummenhofer, Viktor Seidel, Jörg Heckenbach, and Daniel O'Hagan. 2023. Experimental Comparison of Starlink and OneWeb Signals for Passive Radar. In *2023 IEEE Radar Conference (RadarConf23)*. 1–6. https://doi.org/10.1109/RadarConf2351548.2023.10149580

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

[7] Luke Curley, Kirill Pugin, Suhas Nandakumar, Victor Vasiliev, and Ian Swett. 2024. Media over QUIC Transport. https://datatracker.ietf.org/doc/draft-ietf-moq-transport.

[8] DASH Industry Forum. 2023. Dash-Industry-Forum/Dash.Js. https://github.com/Dash-Industry-Forum/dash.js.

[9] DASH Industry Forum. 2023. Dash-Industry-Forum/Livesim2. https://github.com/Dash-Industry-Forum/livesim2.

[10] Jayasuryan V. Iyer, Khasim Shaheed Shaik Mahammad, Yashodhan Dandekar, Ramakrishna Akella, Chen Chen, Phillip E. Barber, and Peter J. Worters. 2022. System and Method of Providing a Medium Access Control Scheduler.

[11] Theo Karagkioules, Rufael Mekuria, Dirk Griffioen, and Arjen Wagenaar. 2020. Online Learning for Low-Latency Adaptive Streaming. In *Proceedings of the 11th ACM Multimedia Systems Conference (MMSys '20)*. Association for Computing Machinery, New York, NY, USA, 315–320. https://doi.org/10.1145/3339825.3397042

[12] Jiajia Liu, Yongpeng Shi, Zubair Md. Fadlullah, and Nei Kato. 2018. Space-Air-Ground Integrated Network: A Survey. *IEEE Communications Surveys & Tutorials* 20, 4 (2018), 2714–2741. https://doi.org/10.1109/COMST.2018.2841996

[13] Yanmei Liu, Yunfei Ma, Quentin De Coninck, Olivier Bonaventure, Christian Huitema, and Mirja Kühlewind. 2023. Multipath Extension for QUIC. https://datatracker.ietf.org/doc/draft-ietf-quic-multipath-06.

[14] Nitinder Mohan, Andrew Ferguson, Hendrik Cech, Rohan Bose, Prakita Rayyan Renatin, Mahesh Marina, and Jörg Ott. 2024. A Multifaceted Look at Starlink Performance. In *Proceedings of the ACM Web Conference 2024 (WWW '24)*. Association for Computing Machinery, New York, NY, USA, 12 pages. https://doi.org/10.1145/3589334.3645328

[15] Owens Nathan. 2023. Unoffical Starlink Global Gateways & PoPs. https://tinyurl.com/nathanstarlink.

[16] Netflix. 2023. Netflix/Vmaf. https://github.com/Netflix/vmaf.

[17] Piers O'Hanlon and Adil Aslam. 2023. Latency Target Based Analysis of the DASH.Js Player. In *Proceedings of the 14th Conference on ACM Multimedia Systems (MMSys '23)*. Association for Computing Machinery, New York, NY, USA, 153–160. https://doi.org/10.1145/3587819.3590971

[18] Jianping Pan, Jinwei Zhao, and Lin Cai. 2023. Measuring a Low-Earth-Orbit Satellite Network. In *2023 IEEE 34th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC) (PIMRC '23)*. 1–6. https://doi.org/10.1109/PIMRC56721.2023.10294034

[19] Crist Ry and Paul Trey. 2023. Starlink Internet Explained. https://www.cnet.com/home/internet/starlink-satellite-internet-explained/.

[20] SpaceX. 2023. Starlink. https://www.starlink.com.

[21] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. 2019. From Theory to Practice: Improving Bitrate Adaptation in the DASH Reference Player. *ACM Transactions on Multimedia Computing, Communications, and Applications* 15, 2s (July 2019), 67:1–67:29. https://doi.org/10.1145/3336497

[22] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. 2020. BOLA: Near-Optimal Bitrate Adaptation for Online Videos. *IEEE/ACM Transactions on Networking* 28, 4 (Aug. 2020), 1698–1711. https://doi.org/10.1109/TNET.2020.2996964

[23] Emily Strong, Bernard Kleynhans, and Serdar Kadıoğlu. 2021. MABWISER: Parallelizable Contextual Multi-armed Bandits. *International Journal on Artificial Intelligence Tools* 30, 04 (June 2021), 2150021. https://doi.org/10.1142/S0218213021500214

[24] Hammas Bin Tanveer, Mike Puchol, Rachee Singh, Antonio Bianchi, and Rishab Nithyanand. 2023. Making Sense of Constellations: Methodologies for Understanding Starlink's Scheduling Algorithms. In *Companion of the 19th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT '23)*. Association for Computing Machinery, New York, NY, USA, 37–43. https://doi.org/10.1145/3624354.3630586

[25] The Pyodide development team. 2023. Pyodide/Pyodide. https://github.com/pyodide/pyodide.

[26] Yufei Wang, Lin Cai, and Jun Liu. 2023. High-Reliability, Low-Latency, and Load-Balancing Multipath Routing for LEO Satellite Networks. In *2023 Biennial Symposium on Communications (BSC)*. 107–111. https://doi.org/10.1109/BSC57238.2023.10201829

[27] Haoyuan Zhao, Hao Fang, Feng Wang, and Jiangchuan Liu. 2023. Realtime Multimedia Services over Starlink: A Reality Check. In *Proceedings of the 33rd Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV '23)*. Association for Computing Machinery, New York, NY, USA, 43–49. https://doi.org/10.1145/3592473.3592562

[28] Jinwei Zhao. 2024. LENS: A LEO Satellite Network Measurement Dataset. https://github.com/clarkzjw/LENS.

A ARTIFACT APPENDIX

A.1 Abstract

This artifact contains the source files for our modified dash.js [8] player and a custom end-to-end LLL video streaming framework for scheduling and running experiments with both network emulation and real-world networks.

Additionally, this artifact contains the network measurement data and necessary scripts to generate the figures in Section 3. It also contains the experiment configuration files and Dockerfiles to build the Docker images for running the experiments in Section 5.

Finally, this artifact contains the results of our LLL video streaming experiments, including all three scenarios, namely *Emulation*, *Starlink* and *Terrestrial*, and the corresponding scripts to generate the figures in Section 5.

A.2 Artifact check-list (meta-information)

- **Algorithm:** Dynamic, L2A-LL, LoL+, CMAB
- **Data set:** CTA WAVE Test Project [1]
- **Run-time environment:** Debian-based Linux distribution with Docker installed (Debian 12 recommended)
- **Hardware:** x86-64 bare-metal Linux server or virtual machine
- **Output:** LLL video streaming metrics captured from the modified dash.js player
- **How much disk space required (approximately)?:** 128 GB
- **How much time is needed to prepare workflow (approximately)?:** Around 1 hour
- **How much time is needed to complete experiments (approximately)?:** Around 14 hours if running the full set of experiments
- **Publicly available?:** Yes
- **Code licenses (if publicly available)?:** GPL-3.0
- **Data licenses (if publicly available)?:** CC BY-SA 4.0
- **Archived (provide DOI)?:** 10.24433/CO.7355266.v1

A.3 Description

A.3.1 How to access. All source files, configurations and scripts are available at the GitHub repository: https://github.com/clarkzjw/mmsys24-starlink-livestreaming. The README.md file provides an overview of the repository structure.

A.3.2 Hardware dependencies. Any bare-metal Linux server or virtual machine capable of running recent versions of Docker and Docker Compose should suffice. For network emulation and real-world experiments, a machine with at least 16 GB of RAM and 4 CPU cores is recommended.

A.3.3 Software dependencies. The results in this paper were produced on Debian 12.5 x86-64 with Docker version 25.0.3 and the Docker Compose plugin version 2.24.5. Only Debian-based Linux distributions are tested.

A.4 Re-generate paper results

To re-generate the figures in the paper, users can directly use the code capsule published on Code Ocean: https://doi.org/10.24433/CO.7355266.v1, and click the "Reproducible Run" button. It contains the necessary software dependencies pre-installed and the figures will be generated automatically in the "results" folder. Alternatively, users can follow the steps in the README.md file and run the provided Docker image to generate the figures. The figures will be generated in the "paper-figures" folder.

A.5 Network emulation and real-world experiments

A.5.1 Installation. To replicate the experiments using either network emulation or real-world networks, or to conduct custom experiments utilizing our end-to-end LLL video streaming framework, users are encouraged to follow the steps in the README.md file for detailed installation instructions.

A.5.2 Evaluation and expected results. The complete suite of experiments, available in the `experiments` folder in the repository, is expected to take approximately 14 hours to complete with either network emulation or real-world networks. Afterward, the results can be found in the `figures` folder. A helper Docker image is provided to quickly generate all the figures. See the README.md file in the GitHub repository for more details.

A.6 Experiment customization

The video streaming experiments can be customized by modifying the configuration files in the `experiments` folder. The following parameters can be adjusted:

- **EMULATION**: This option should be enabled for users intending to conduct the experiments with network emulation.
- **ROUND_DURATION**: The time duration of each round in seconds.
- **TARGET_LATENCY**: The target latency in seconds.
- **CONSTANT_VIDEO_BITRATE**: When set to `-1`, ABR algorithm is effective. Otherwise, the dash.js player uses the fixed video bitrate as specified in Kbps.
- **CMAB_ALPHA**: The exploration parameter for the CMAB algorithm.
- **TOTAL_ROUNDS**: The total number of rounds to be repeated for the experiment.
- **MPD_URL**: The URL of the MPD manifest file. When using the provided Docker images in network emulation mode, this option should not be changed.
- **ABR_ALGORITHM**: The name of the ABR algorithm. The following options are available: *abrDynamic*, *abrL2A*, *abrLoLP* and *abrCMAB*, where *abrCMAB* is the proposed algorithm in the paper. Note that these are internal names, they map to the *Dynamic*, *L2A-LL*, *LoL+* and the proposed *CMAB*-based ABR algorithm in the paper, respectively.
- **CATCH_UP**: The name of the catch-up policy. The following options are available: *liveCatchupModeDefault*, *liveCatchupModeLoLP* and *liveCatchupModeCMAB*, where *liveCatchupModeCMAB* is the proposed catch-up policy in the paper. Note that these are internal names, they map to the *Default*, *LoL+* and the proposed handover-aware catch-up policy in the paper, respectively.