StarTCP: Handover-aware Transport Protocol for Starlink

Li Jiang[1], Yihang Zhang[1], Yannan Hu[2], Yong Cui[3], Xinggong Zhang[1]
[1] Peking University, [2] Zhongguancun Laboratory, [3] Tsinghua University

ABSTRACT

Legacy transport protocols such as TCP and QUIC suffer from high packet loss and low link utilization in Starlink. From the measurement data, we figure out the ground-satellite link (GSL) handover is mainly to blame. The periodic handovers result in link interruptions and bursty losses with a fixed interval of 15*s*, which impair TCP's performance. Based on this finding, we present a handover-aware transport protocol, StarTCP, which proactively stalls transmission during handovers to avoid bursty losses and erroneous congestion signals. Preliminary results indicate that StarTCP can efficiently reduce packet loss and enhance throughput in Starlink.

CCS CONCEPTS

• **Networks → Transport protocols**.

ACM Reference Format:
Li Jiang[1], Yihang Zhang[1], Yannan Hu[2], Yong Cui[3], Xinggong Zhang[1]. 2024. StarTCP: Handover-aware Transport Protocol for Starlink. In *The 8th Asia-Pacific Workshop on Networking (APNet 2024), August 03–04, 2024, Sydney, Australia.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3663408.3665803

1 INTRODUCTION

As a notable leader of Low Earth orbit (LEO) constellations, Starlink has served more than 2.3 million users in over 70 countries. However, the surging measurement studies [2, 5] indicate that in today's transport layer, TCP and QUIC suffer from high packet loss rate (PLR) and low bandwidth utilization in Starlink. The packet loss is as high as over 0.4%[5], and the throughput even with the state-of-the-art BBR is as low as half the bandwidth[2], which is noticeably lower than that of the terrestrial network.

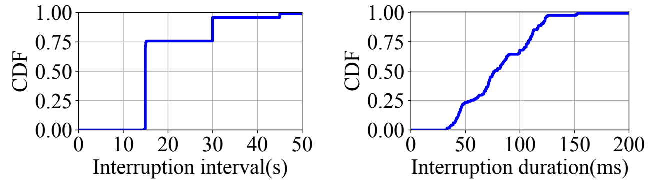
Based on Starlink traffic dataset [5], we conclude that the **periodic handover of ground-satellite links (GSL)** is to blame. The handovers trigger non-negligible link interruptions periodically. Figure 1a shows that for the interruption events [1], **the intervals are consistently at 15x seconds**. This echos the claim [1] that Starlink plans handovers on a fixed 15-second interval, indicating that **the periodic interruptions are caused by handovers**. Figure 1b displays the durations of these interruptions. It is noticed that the duration is randomly distributed from 30ms to 200ms.

[1]Loss events with duration exceeding 11ms. According to the equation in [4], a handover in Starlink brings an interruption of at least 11ms.

This work was sponsored by the NSFC grant (U21B2012) and Huawei Grant (TC20230914045). Xinggong Zhang is the corresponding author.

APNet 2024, August 03–04, 2024, Sydney, Australia

ACM ISBN 979-8-4007-1758-1/24/08
https://doi.org/10.1145/3663408.3665803



(a) interval **(b) duration**

Figure 1: The intervals and durations of interruption events.

The regular link interruptions bring bursty packet loss frequently. The transport protocols are prone to interpret these losses as congestion signals and improperly decrease the congestion window, leading to low throughput and high queuing delays.

Furthermore, handovers also cause **long transmission recovery times**. During the handover, ACKs are also lost and the sending window is blocked as none of the packets within it are acknowledged. Even when the handover is over, new packets cannot be sent since the sending window remains filled with these unACKed packets. Meanwhile, the receiver will not issue ACKs until receiving new packets. It takes a longer time than the handover itself to resume transmission, also contributing to low link utilization.

Previous works [1, 3] to address the issue of handovers use additional information for handover prediction, which is impractical in Starlink. They also underestimate the interruption duration and just optimize for loss handling, thus cannot prevent sending blocking.

Our insight is that **stalling packet transmission during handovers** can be effective. First, this prevents bursty packet loss, thereby eliminating erroneous congestion signals and preventing incorrect decreases of the congestion window. Second, it helps avoid sending blocking. As packet emission is stopped during handovers, the sending window would not be filled. New packets can be sent immediately once the link is recovered.

However, the design is not trivial. There are three crucial technique challenges. (i) **How to predict the occurrence of handovers?** Although handovers are periodic, accurately predicting their timing remains challenging. The only information we can rely on is historical traffic. (ii) **How to empty the inflight packets?** When a handover occurs, there are still ongoing packets in the network. Losing them can also trigger incorrect congestion signals. (iii) **How to quickly detect link recovery?** The interruption duration of a handover is quite random, making it difficult to anticipate when the transmission could be resumed.

We present StarTCP, a transport protocol specially designed for Starlink to address these challenges. (1) It employs a statistics-based algorithm to predict future handovers with historical traffic. It detects interruption events by monitoring abnormal packet intervals and then identifies the periodic handovers leveraging the fixed handover interval. (2) The sending is stalled proactively ahead of handovers to prevent the loss of inflight packets. It utilizes reinforcement learning (RL) to learn the optimal stalling time under various network conditions. (3) A link probing mechanism is proposed. The probe packets continuously sent during handovers enable StarTCP to promptly detect link recovery and resume sending.

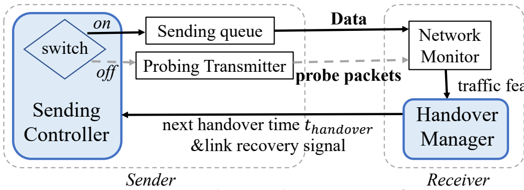


Figure 2: The architecture of StarTCP.

2 DESIGN

Figure 2 shows the two key modules in StarTCP: the *Handover Manager* at the receiver, and the *Sending Controller* at the sender.

The *Handover Manager* detects the handover status with traffic features of received packets. It notifies the sender when it predicts the next handover time. Upon perceiving link recovery after the handover, it informs the sender to resume transmission.

The *Sending Controller* determines the sending mode at any given moment. When the switch is on, the sender sends packets from its sending queue as usual. The switch will be turned off when a handover is imminent. The data transmission is suspended and the sender starts to send probe packets. In this way, bursty losses during handovers are avoided. The probe packets enable the receiver to promptly detect link recovery as soon as the handover is over.

2.1 Handover prediction

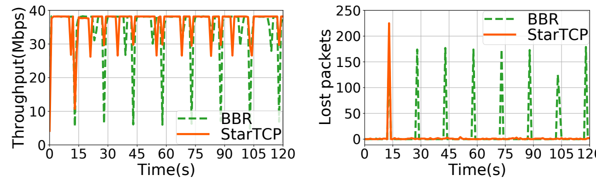
StarTCP predicts the next handover time based on historical traffic. Our insight is that: by leveraging the fixed interval, we can predict future handovers if the preceding ones can be detected. The *Handover Manager* identifies a series of interruptions by monitoring abnormal packet intervals, which could potentially indicate handover events. We denote the start time of these interruptions as $t_1, t_2, ..., t_n$. Our target is to compute the next handover time $t_{handover}$ with knowing that the handover interval T is 15 seconds.

We employ **kernel density estimation (KDE)** to distinguish the handover-related interruptions from the others. The core idea is that: by dividing the interruption times by T, the interruptions caused by handovers are mapped to almost the same residue. Therefore we identify the point with the highest density among these residues as the base time of handovers. The next handover time must be multiple handover intervals after the base time.

2.2 Transmission stalling strategy

With having $t_{handover}$, the *Sending Controller* should stop sending data in advance to empty the inflight packets before $t_{handover}$. In LEO satellite networks, the link condition varies over time and the optimal strategy is different for different network environments. This is because the arrival time of ongoing packets depends on the future network conditions. When the bandwidth drops suddenly, the packets are delayed and can be lost in the handover. We prefer to stall transmission more conservatively in highly variable network conditions to avoid packet loss, while stalling too early in a stable network environment results in unnecessary bandwidth wastage. Therefore, StarTCP use an **actor-critic model to learn how far before the handover should the sending be stopped** under various network conditions. The specific functionalities are:

(i) *State.* The network takes the state $s_t = (\vec{x}_t, \vec{v}_t, \vec{b}_t, d, r)$ as input. Here, $\vec{x}_t$, $\vec{v}_t$, $\vec{b}_t$ represent the mean RTTs, mean RTT variances and data acknowledgment rates of the past n time intervals respectively. d denotes the estimated OWD computed by a heuristic algorithm and r indicates the remaining time until next handover $t_{handover}$.



(a) throughput **(b) lost packets**

Figure 3: Trace of throughput and packet loss.

(ii) *Reward.* After each handover, the agent evaluates its performance during that period. Aimed at achieving high throughput with low packet loss, the reward function is defined as $R_t = \alpha \cdot u_t - \beta \cdot l_t$, where u_t and l_t represent the throughput and PLR in a 1-second window that contains the handover.

(iii) *Offline training*. Since the agent receives feedback and updates its model every 15s, the model is trained offline in various network environments with different settings of delay, jitter, bandwidth, and bandwidth variance.

2.3 Link probing mechanism

To detect link status without experiencing losses, probe packets are utilized during handovers. When the transmission is stopped, the *Probing Transmitter* continuously generates lightweight probe packets and sends them into the network at a 1-ms interval. First, they help to provide *HandoverManager* the accurate start time of each interruption. And Upon receiving the first probe packet after the interruption, the receiver promptly perceives link recovery and notifies the sender to stop probing and resume sending data.

3 PRELIMINARY RESULTS

Figure 3 shows how StarTCP works through an example case. The path switches between two parallel links every 15s with an interruption of 100ms to simulate handovers in Starlink. Both links have an RTT of 100ms and a bandwidth of 40Mbps. Although BBR achieves optimal performance for most of the time, its throughput drops to 6Mbps with more than 150 packets lost during the second when the handover occurs. Using BBR as its congestion control algorithm, StarTCP initially behaves similarly to BBR in the first handover at the 13th second. However, in the subsequent handovers, StarTCP increased its throughput to 30Mbps with almost no packet loss, as it has already been able to predict handovers and react proactively.

REFERENCES

[1] 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

[2] Mohamed M. Kassem, Aravindh Raman, Diego Perino, and Nishanth Sastry. 2022. A browser-side view of starlink connectivity. In *Proceedings of the 22nd ACM Internet Measurement Conference* (Nice, France) *(IMC '22)*. Association for Computing Machinery, New York, NY, USA, 151–158. https://doi.org/10.1145/3517745.3561457

[3] Xiangyu Li and Guixian Wang. 2022. Handover Detection and Fast Recovery over Satellite Networks for WebRTC. In *2022 IEEE 8th International Conference on Computer and Communications (ICCC)*. 599–603. https://doi.org/10.1109/ICCC56324.2022.10065763

[4] Helka-Liina Maattanen, Bjorn Hofstrom, Sebastian Euler, Jonas Sedin, Xingqin Lin, Olof Liberg, Gino Masini, and Martin Israelsson. 2019. 5G NR Communication over GEO or LEO Satellite Systems: 3GPP RAN Higher Layer Standardization Aspects. In *2019 IEEE Global Communications Conference (GLOBECOM)*. 1–6. https://doi.org/10.1109/GLOBECOM38437.2019.9014090

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