LEOTP: An Information-centric Transport Layer Protocol for LEO Satellite Networks

Li Jiang*, Yihang Zhang*, Jinyu Yin*, Bin Liu†, Xinggong Zhang*
* Wangxuan Institute of Computer Technology, Peking University, Beijing, China
† Department of Computer Science and Technology, Tsinghua University, Beijing, China
{jl99888, zhangyihang, yinjinyu, zhangxg}@pku.edu.cn, {liub}@tsinghua.edu.cn

***Abstract*—Low Earth orbit (LEO) satellite networks have attracted extensive research due to their potential to provide high-quality Internet access services. However, the existing TCP variants, which are designed for terrestrial networks, can hardly work in LEO satellite networks with characteristics such as error-prone, bandwidth variations, and link switching. To address these challenges, in this paper we present a new information-centric transport layer protocol LEOTP to guarantee reliable, high-throughput, and low-latency data transmission in LEO satellite networks. It leverages the idea of Information-Centric Networking (ICN) with a Request-Response transmission model and in-network caching. The connectionless transmission paradigm in LEOTP makes it resilient to dynamic topology changes. The caches equipped in intermediate nodes help to recover packet loss while the hop-by-hop congestion control mechanism provides a fast reaction to time-varying network conditions. We evaluate the performance of LEOTP in emulated Starlink constellation, which shows that it increases the throughput by 8%-12% with 40%-60% delay reduction compared with the state-of-the-art TCP variants in the transcontinental data transmission.**

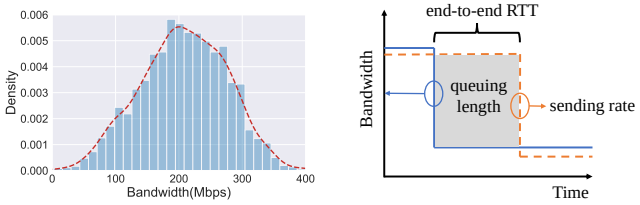
***Index Terms*—LEO, satellite networks, transport layer protocol, ICN, segmented transmission control**

I. Introduction

In recent years, low Earth orbit (LEO) satellite networks are fast emerging. As of 2 December 2022, SpaceX has launched 3,558 Starlink satellites [1]. OneWeb [2], Amazon [3], Telesat [4], and more players are entering the market. As Starlink's slogan says, the LEO satellite network aims to provide "High speed, Low latency broadband connectivity across the globe". Compared with the terrestrial network, the LEO satellite network could easily cover wide areas at a low cost, while compared with the geosynchronous-equatorial-orbit (GEO) satellite network it has much lower latency. Due to these inspiring features, the LEO satellite network is regarded as an important part of the next-generation networks called Space-Ground Integrated Network (SGIN).

One urgent issue is emerging: *While the highway is ready, where is the safety belt?* It's well-known that in today's Internet, TCP is the *safety belt* to provide reliable end-to-end transmission. However, TCP is designed for terrestrial networks, its performance is poor in the error-prone, highly-dynamic LEO satellite network.

The recent measurement study on Starlink [5] indicates that the packet loss rate (PLR) and the latency are high in LEO satellite networks. First, although inter-satellite links (ISL) are not currently enabled, the PLR on ground-satellite links (GSL) is 1.56% for downloads and 1.96% for uploads [5]. This deteriorates the throughput of the loss-based TCP variants. Besides, the lost packets have an extra retransmission delay of at least one Round-Trip Time (RTT), which is large in long-distance communications. Second, for single-flow data transfer from Belgium to European destinations in the Netherlands and Germany, the median and 95th percentile RTT are 95ms and 175ms [5]. This makes TCP hard to support latency-sensitive services. As the propagation delay to those destinations is only at the order of 20ms [5], the rest of the latency is caused by queuing in the network. So the congestion is significant in LEO satellite networks.



(a) Bandwidth distribution in Starlink (b) Queuing due to bandwidth variations

Fig. 1: Bandwidth variations and delayed feedback results in congestion in LEO satellite networks.

The bandwidth variation with delayed feedback is a major reason for the large queuing delay. Fig. 1a shows the distribution of Starlink's download bandwidth according to the data published by [5]. The fast movement of LEO satellites results in frequent handover and routing switching, which makes the connection intermittent [6]. The link quality and connectivity of GSLs are also highly-related to the weather condition [7]. Therefore, the bottleneck bandwidth in LEO satellite networks is time-varying, ranging from 2Mbps to 386Mbps. Meanwhile, TCP relies on end-to-end feedback control. The congestion signals are carried in ACKs echoed by the receiver. As shown in Fig. 1b, when the bandwidth drops suddenly, the sender cannot adjust its sending rate in time with the long feedback cycle of an end-to-end RTT. Then the packets start to queue at the bottleneck, which increases the latency and causes packet loss when the queue is filled up.

Our key insight is that these problems can be solved by segmented transmission control. Under this scheme, retransmission and congestion control are performed at each individ-

ual hop. First, the in-network retransmission enables packet loss to be detected and repaired locally, which reduces the recovery time and bandwidth consumption of retransmissions. Second, the hop-by-hop congestion control has a lag time much shorter than the end-to-end RTT. So it can react quickly to bandwidth variations, achieving lower latency and higher throughput accordingly.

However, the design of segmented transmission control is not trivial. Split TCP [8]–[10] is a well-known existing solution. It uses proxies to break the end-to-end connection into independent TCP connections by hop. But this straightforward method does not work in LEO satellite networks. There are three crucial technical challenges:

(i) **How to keep the connection in dynamic topology?** Split TCP builds a connection on each hop. However, the connection state kept by an intermediate node is lost when the node is moved away, which interrupts data transfer.

(ii) **How to achieve end-to-end reliability?** Split TCP provides per-hop reliability, while end-to-end reliability is not guaranteed. When an intermediate node removes from the path in link switching, the packets buffering on it are lost and will not be repaired. Therefore, the end-to-end reliability is broken.

(iii) **How to avoid backlog at intermediate nodes?** If each hop controls traffic independently, the queue will build up at intermediate nodes due to the bandwidth difference between adjacent hops, resulting in even higher end-to-end latency.

In this paper, we propose an information-centric transport layer protocol LEOTP to address these challenges. It leverages the ideas of the Request-Response model and in-network caching from Information-Centric Networking (ICN). The end receiver issues data requests and waits for data to be returned. Any node, i.e. data source or an intermediate node with cache, can respond with the requested data. The receiver re-issues a request for data not received with a timeout mechanism. The intermediate nodes detect packet loss by the sequence number of received packets and retransmit locally. The congestion control is performed at each hop. First, under the pull-based transmission, only the receiver records the states of ongoing packets. The intermediate nodes are dummy nodes, so LEOTP is resilient to the dynamic topology. Second, LEOTP has a hybrid retransmission mechanism. The receiver-driven retransmission provides end-to-end reliability, while the retransmission on each hop minimizes the delay and bandwidth consumption of data recovery. Third, to avoid lost packets to be detected repeatedly by intermediate nodes, LEOTP uses the Void Packet Header (VPH) mechanism. Packet headers are sent to downstream nodes as notifications when detecting packet loss. Fourth, LEOTP combines a backpressure algorithm with the hop-by-hop congestion control to coordinate the sending rate between hops. So the packet backlog at intermediate nodes is avoided.

We implement LEOTP based on UDP, without requiring modifications to the lower layers. We carry out extensive experiments to evaluate its performance. Compared with the state-of-the-art TCP variants, (1) in the controlled experiments, LEOTP achieves 12% higher throughput under high PLR and 46% latency reduction under bandwidth fluctuations. (2) LEOTP has good fairness between flows with different RTTs. (3) In the Starlink emulation experiments, LEOTP increases end-to-end throughput by up to 8% with 40% lower latency on the Beijing-New York link. (4) To save costs, the intermediate nodes can be deployed on part of the satellites. LEOTP can achieve 6% higher throughput and 42% lower latency at only 25% coverage in the same environment.

In summary, our specific contributions are:

- A discussion of the limitations of end-to-end transport protocols in LEO satellite networks.
- An information-centric, cache-assisted transport protocol, called LEOTP, achieves reliability, high throughput, and low latency in LEO satellite networks. The code and key results in this paper are publicly available [11].
- A novel in-network retransmission mechanism using VPH as notifications, which reduces redundant retransmissions while providing fast loss recovery.
- A backpressure-based hop-by-hop congestion control that provides quick reactions in long-distance networks.
- An evaluation of LEOTP in a large-scale emulated constellation with multiple ground stations and satellites.

II. Motivation and Chance

In this section, we will analyze the problems of the existing methods, and discuss what is required for a transport protocol that fits LEO satellite networks.

A. TCP's Performance Degradation in LEO Satellite Networks

To demonstrate how high PLR and bandwidth variations in LEO satellite networks degrade the performance of TCP, we perform a set of simulation experiments. The link parameters are chosen close to the real network environment.

High PLR degrades the throughput of TCP. Besides packet loss on GSLs, recent studies [12], [13] suggest that ISLs are also error-prone with PLR of 1%. This implies that once ISLs are enabled, the end-to-end PLR will grow proportionally with the hop count and reach up to 5%. In Fig. 2, we set each hop with 20Mbps bandwidth, 10ms RTT, and 0.5% PLR. The throughput of the loss-based congestion control algorithms Cubic [14] and Hybla [15] decreases dramatically to less than 2Mbps when the hop count is 5. This is because all loss events are ascribed to congestion, causing an unnecessary decrease of congestion window. The state-of-the-art algorithms BBR [16] and PCC [17] are loss-insensitive, so their throughput degradation due to packet loss is not severe when the hop count is small. However, when the hop count increases to 10 with the end-to-end PLR of 5%, the throughput of BBR and PCC also decreases by 9% and 33% respectively. For BBR, an important reason is that the lost packets are retransmitted over the entire link, taking up a non-negligible amount of the bottleneck bandwidth. Meanwhile, although PCC is not loss-based, it is also not robust enough against high PLR.

High PLR results in a high tail latency for TCP. The data retrieval delay of those lost packets is at least one RTT larger than normal packets. This is due to the fact that TCP uses an

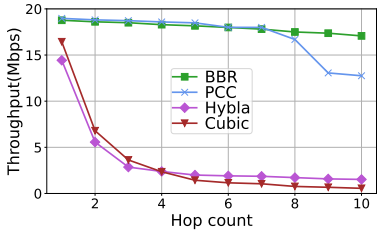


Fig. 2: The throughput degradation in error-prone links.

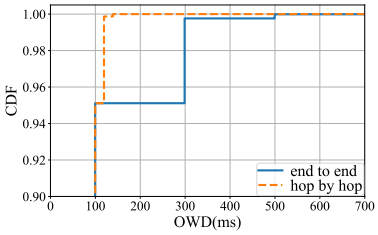


Fig. 3: Theoretical per-packet OWD distribution under end-to-end and hop-by-hop retransmission.

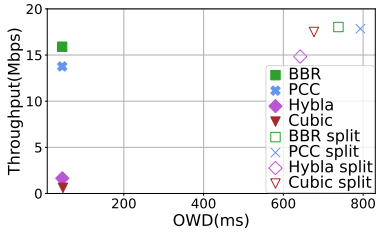
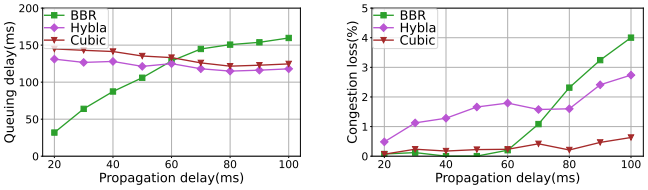


Fig. 4: Throughput-OWD trade-off for Split TCP and TCP



(a) Queuing delay (b) Congestion loss

Fig. 5: The queuing delay and congestion loss under different propagation delay in bandwidth fluctuations.

end-to-end retransmission mechanism, which means the packet loss is not detected until the data reaches the receiver and can only be repaired by the sender. Fig. 3 shows the theoretical one way delay (OWD) distribution in a 10-hop network, each hop has 0.5% PLR and 10ms delay. We can see there is a long tail under TCP's end-to-end retransmission. Of the 100000 packets we simulate, the 99th percentile and the maximum OWD are 300ms and 700ms respectively. This long tail makes TCP hard to be compatible with latency-sensitive applications.

TCP has a high queuing delay under bandwidth variations. Fig. 5 shows the queuing delay and congestion loss under different propagation delays. The average bottleneck bandwidth is 10Mbps and fluctuates as a square wave with a fixed period (2s) and amplitude (1Mbps), the bandwidth of all other segments is 20Mbps. BBR is a typical delay-bounding algorithm. It periodically probes the bandwidth and adjusts its sending rate accordingly, aiming to work at an empty queue and provide low latency. The loss-based Cubic and Hybla are presented as references. As the maximum RTT between city pairs will be over 150ms with ISLs [18], the maximum propagation delay is set to 100ms. In Fig. 5a, Cubic and Hybla maintain a stable but large queuing delay of over 100ms because they keep increasing the congestion window until the bottleneck buffer is full. When the propagation delay is 20ms, BBR has an average queuing delay of 32ms. As the feedback cycle increases with the propagation delay, its queuing delay increases constantly and can be even higher than the loss-based algorithms. This is because BBR has a longer lag time to probe the change of bandwidth. It fails to adjust the sending rate promptly, leading to severe congestion. In Fig. 5b, the congestion loss increases with the propagation delay for all the algorithms, as they take a longer time to react when congestion occurs, causing more packets to overflow from buffers.

B. Potential Improvement of Segmented Transmission Control

We validate that segmented transmission control has certain advantages in LEO satellite networks. First, it reduces the cost of retransmission. When enabling in-network retransmission, the lost packets can be retransmitted locally on each hop. For a network of N hops, each of which has PLR p, propagation delay d, and bandwidth b, the end-to-end PLR P is as (1):

$$P = 1 - (1-p)^N \approx Np \tag{1}$$

On the one hand, the recovery delay is shortened from the end-to-end RTT to only one hop RTT (hopRTT). So the average OWD under end-to-end and hop-by-hop retransmission are calculated as (2), (3), where k is the retransmission times.

$$\overline{OWD_{ete}} = \sum_{k=0}^{+\infty}(1+2k)Nd(1-P)(P)^k \approx Nd \cdot \frac{1+Np}{1-Np} \tag{2}$$

$$\overline{OWD_{hbh}} = N\sum_{k=0}^{+\infty}(1+2k)d(1-p)(p)^k = Nd \cdot \frac{1+p}{1-p} \tag{3}$$

On the other hand, the extra bandwidth consumption for retransmission is limited to one hop. Specifically, for packet loss that occurs on non-bottleneck links, the retransmission does not occupy the bottleneck bandwidth. So the theoretical upper bound of throughput under end-to-end and hop-by-hop retransmission can be calculated as (4) and (5) respectively:

$$Throughput_{ete} = \frac{b}{\sum_{k=0}^{+\infty} P^k} \approx b(1-Np) \tag{4}$$

$$Throughput_{hbh} = \frac{b}{\sum_{k=0}^{+\infty} p^k} = b(1-p) \tag{5}$$

So compared with end-to-end retransmission, hop-by-hop retransmission has $\frac{1-p}{1-Np}$ times the theoretical throughput and $\frac{(1+p)(1-Np)}{(1-p)(1+Np)}$ times the average OWD. Taking an example, when $N = 10, p = 0.5\%$, hop-by-hop retransmission achieves 4.7% higher theoretical throughput and 8.7% lower average OWD. Meanwhile, the tail latency obtains a more

significant improvement. The OWD distribution under hop-by-hop retransmission is also plotted in Fig. 3. The 99th percentile and the maximum OWD decrease to 120ms and 160ms respectively, which efficiently mitigate the long tail.

Second, **segmented transmission control has a faster reaction to the varying bandwidth** due to its reduced feedback loop. When the bandwidth drops instantly, any node can sense the congestion from the feedback provided by its neighboring nodes. So it can make adjustments rapidly to avoid congestion. As a result, the queue generated by overshoot is much shorter than that of end-to-end congestion control. And when the bandwidth increases, it can also probe the spare bandwidth immediately and increase its sending rate. This indicates that hop-by-hop congestion control has the potential to achieve both lower latency and higher throughput under the bandwidth variations in LEO satellite networks.

Third, segmented transmission control improves the end-to-end throughput under high PLR. We prove this by combining Split TCP with different congestion control algorithms, as shown in Fig. 4. The experiment is performed in a 10-hop network, with 20Mbps bandwidth, 10ms RTT, and 0.5% PLR on each hop. Compared to the end-to-end link, each hop has a better link quality with a much lower PLR and RTT. So Cubic and Hybla reduce the congestion window less frequently and their throughput increases significantly from less than 2Mbps to over 14Mbps. Meanwhile, BBR and PCC also have 13% and 30% throughput improvement when being split.

C. Design Challenges

While we are motivated to enable segmented transmission control in LEO satellite networks, we notice that the straightforward method of split TCP faces several problems. We summarize these as the design challenges for our proposal.

Support for mobility. To enlarge the benefits of segmented transmission control, it is necessary to use LEO satellites as intermediate nodes. However, Split TCP establishes a connection for each hop during the handshake between the sender and the receiver, and the connection states are kept by the intermediate nodes. In LEO satellite networks, a satellite can be moved away in a short time. In this case, the endpoints can not perceive that it has been removed from the path and will not re-establish the connections. Its upstream node still tries to forward data to it. So all the packets are dropped and the end-to-end data transfer is interrupted.

End-to-end reliability. The reliability of Split TCP is provided by retransmission at each hop. However, even if the transmission at each individual hop is completely reliable, end-to-end reliability is not guaranteed. The key reason is that packet loss may occur on the intermediate nodes rather than on a hop. This can happen when an intermediate node removes from the path due to link switching. The packets in its buffer are lost and can not be repaired. This is because the upstream node believes the data is delivered correctly as it has been ACKed, and the downstream nodes are unaware of the data.

Packet backlog at intermediate nodes. Split TCP performs independent flow control for each hop. Each node treats its downstream node as the actual receiver, sending as much data as possible regardless of the throughput of the next hop. The throughput difference between adjacent hops causes data to accumulate in the intermediate nodes' buffers, resulting in extremely high latency. We also present the latency of Split TCP in Fig. 4. For all the algorithms, splitting brings an extra queuing delay at intermediate nodes of more than 600ms.

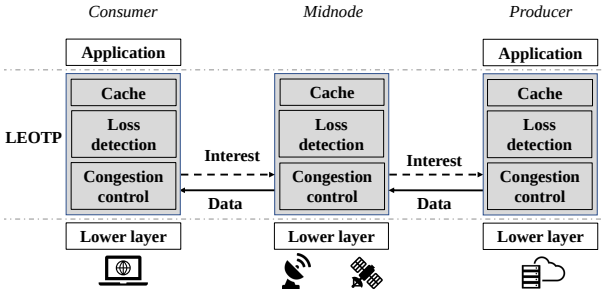


Fig. 6: The overview of LEOTP architecture.

III. Design

In this section, we present the architecture of LEOTP and introduce our design for the two key modules: retransmission and congestion control.

A. System Overview

Fig. 6 shows the architecture of LEOTP. A path consists of a *Consumer*, a *Producer*, and several *Midnode*s. The end receiver is the Consumer, whose application layer finally "consumes" the data. The data source is the Producer, whose application layer "produces" the data. The ground stations and LEO satellites are Midnodes, which refer specifically to intermediate nodes in LEOTP. They have no application layer and are used to enhance the performance of LEOTP. The transport layer is segmented hop-by-hop by the Midnodes. To enable transmission control at each hop, the *Loss detection* module and the *Congestion control* module are deployed at every nodes. Each node is also equipped with a *Cache* to provide local storage for data packets.

The transmission in LEOTP follows the **Request-Response model**. With the information-centric paradigm, each piece of data has a location-independent name. It is composed of the *FlowID* (i.e., the unique identifier of a flow) and its byte-level range in the flow. When a Consumer wants to fetch a piece of data from a Producer, it sends a data request (Interest) to the Producer by its name, then waits for the data. As LEOTP decouples the information of location and naming of data, the requested data can be responded from either the Producer or any hitting cache on the path. So the data transfer in LEOTP is connectionless. The Midnodes are dummy nodes keeping few connection states. They just perform cache lookup and respond data according to the name parsed from the passing Interests. When moved away or disconnected, they do not need to migrate their states to other nodes. In this way, LEOTP supports the mobility of intermediate nodes.

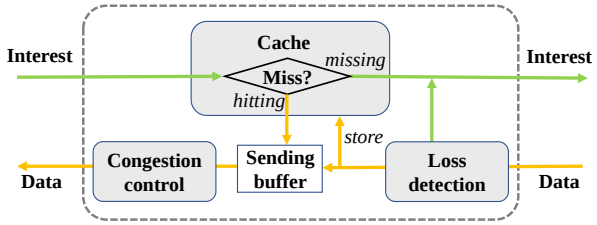
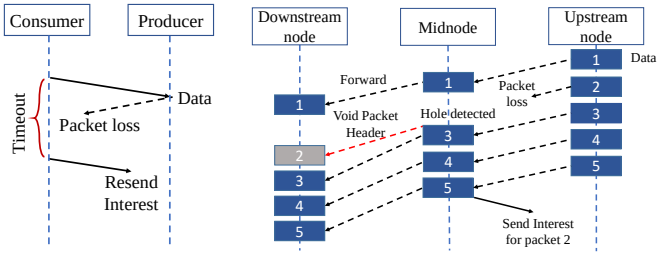


Fig. 7: The key modules in a Midnode.



(a) TR mechanism (b) SHR mechanism

Fig. 8: The hybrid retransmission mechanism.

LEOTP takes the idea of segmented transmission control. The **local retransmission** at each hop aims to minimize the network cost and the **hop-by-hop congestion control** provides a quicker reaction to varying network conditions. We illustrate how they work through Fig. 7. First, the local retransmission is supported by the *Cache* and the *Loss detection* module. The Midnode performs loss detection locally by monitoring the received data. When a packet loss is detected, it sends an Interest to its upstream node for retransmission. Meanwhile, a node stores the data in its *Cache* while forwarding it. When receiving an Interest, a node searches its *Cache* and tries responding to the Interest first. So the lost packet can be recovered from the upstream node immediately. Second, to enable hop-by-hop congestion control, each Midnode has a sending buffer. All the data needs to be forwarded is queued in the sending buffer, and the sending rate is decided by its *Congestion control* module.

B. Reliable Transmission

LEOTP uses a hybrid retransmission mechanism to enable end-to-end reliable transmission. The Consumer-driven retransmission guarantees end-to-end reliability at the last sort, while the in-network retransmission recovers most of the packet loss at a minimal cost of latency and bandwidth consumption. Specifically, the Consumer-driven retransmission is called Timeout Retransmission (TR), which is based on a timeout mechanism. And the in-network retransmission is named as Sequence Hole Retransmission (SHR), as each node detects packet loss according to the sequence numbers of the received data.

TR. The basic idea of TR is shown in Fig. 8a. The Consumer records the sending time of each Interest, and periodically checks all the unsatisfied Interests. If an Interest has not been satisfied by its requested data after Retransmission TimeOut (RTO), it will be resent by the Consumer. The RTO is calculated by the smoothed RTT (SRTT) and the variance of RTT (RTTVAR) like TCP according to the algorithm in RFC6298 [19]. When an Interest has experienced multiple timeouts, the resending interval will grow exponentially by 1.5 each time, until the data arrives.

SHR. In SHR, each node detects packet loss locally by the sequence numbers. When a packet's sequence number is greater than what is expected to be received next, a "hole" appears between this packet and the previously received packets. If the hole is not filled by the following few packets, it is supposed that packet loss has occurred and an Interest is sent to request the corresponding byte range of data.

However, **duplicate retransmission is a potential problem for SHR**. Since the Midnodes just forward data packets without waiting for reordering or retransmission, when a sequence number hole is detected by a Midnode, it will also be detected by all the downstream nodes. If each of these nodes sends an Interest respectively, the lost packet will be retransmitted repeatedly, resulting in a severe waste of bandwidth.

LEOTP uses the Void Packet Header (VPH) mechanism to solve this problem. When a Midnode detects a hole, it will generate a Void Packet Header with the same byte range as the hole and send it downstream immediately. When receiving this header, its downstream nodes are notified that the hole has already been detected by the upstream nodes and then ignore it. In this way, all the downstream nodes will not issue an Interest for retransmission. The retransmission only takes place in the hop where the packet loss occurs. Specially, when the Consumer receives a header, it will reset the timestamp of the corresponding Interest to avoid the timeout being triggered before the data retransmitted by SHR arrives.

Fig. 8b shows an example of SHR. The upstream node sends 5 packets and packet 2 is lost. When having received packet 3, the Midnode detects packet 2 as a hole since the sequence number is not continuous. It then generates a header for packet 2 and sends it downstream. After receiving packet 4 and packet 5, the Midnode confirms that packet 2 is lost and sends an Interest to the upstream node for retransmission. The downstream node, however, has received the header of packet 2 before receiving packet 3. So it perceives continuous sequence numbers and does not regard packet 2 as a lost packet. Then packet 2 is only retransmitted between the Midnode and the upstream node, thus duplicate transmission is avoided.

Algorithm 1 presents the detailed algorithm of SHR. LEOTP is a byte stream protocol. Each data packet carries its byte-level range $[rangeStart, rangeEnd)$. The Midnodes keep track of the largest sequence number seen for a flow, which is named as $lastByte$. When a data packet or a header is received, it is processed according to its range:

(1) *In-sequence packet.* It is the common case when the packet is what is expected to be received next. It is forwarded directly without additional processing.

Algorithm 1 Loss detection algorithm in SHR

Input: the threshold for disordered packets N

$SeqNumHoles \leftarrow \{\}$
while recvData() **do**
  $[rs, re) \leftarrow newPacket.range$
  if $rs > lastByte$ **then**
    generate and send header $[lastByte, rs)$
    insert $[lastByte, rs)$ to $SeqNumHoles$
  else if $rs < lastByte$ **then**
    delete $[rs, re)$ from $SeqNumHoles$
  end if
  for $hole$ in $SeqNumHoles$ **do**
    if $rs > hole.rangeEnd$ **then**
      $hole.count \leftarrow hole.count + 1$
      if $hole.count > N$ **then**
        send Interest for $hole$
        remove $hole$ from $SeqNumHoles$
      end if
    end if
  end for
  $lastByte \leftarrow max(lastByte, re)$
end while

(2) *Out-of-sequence packet whose sequence number is greater than the last received.* It is the case when the packet's $rangeStart$ is greater than the current $lastByte$. A hole is detected, which means its previous few packets have not arrived. They can be lost or delayed by the network. First, the Midnode generates a packet header for the hole, and forwards it before this packet to notify the downstream nodes. To distinguish whether a hole is caused by packet loss or by disordering, LEOTP does not send a retransmission Interest immediately when a hole is detected. Instead, the hole is recorded in $SeqNumHoles$ with $count$ representing the times it is skipped by the following packets. All the received packets update the $count$ of holes in $SeqNumHoles$ according to its byte range. When a hole's $count$ reaches the threshold N, an Interest is issued to request the lost packet. Then the hole is deleted as SHR does not keep track of whether the retransmission is successful or not.

(3) *Out-of-sequence packet whose sequence number is less than the last received.* It is the case when the packet's $rangeStart$ is less than the current $lastByte$. The Midnode has already received some of the following packets. The packet may be a retransmitted packet or a disordered packet delayed by the network. The Midnode will search its $SeqNumHoles$ and delete the holes that overlap with its range, as the data has already arrived and does not need to request again.

C. Backpressure Congestion Control

LEOTP performs hop-by-hop congestion control, combined with a backpressure mechanism to coordinate the sending rate between hops. The basic idea of backpressure is: if the downstream sending rate is lower than the upstream, the upstream will decrease its sending rate actively to avoid packet backlog at the intermediate node.

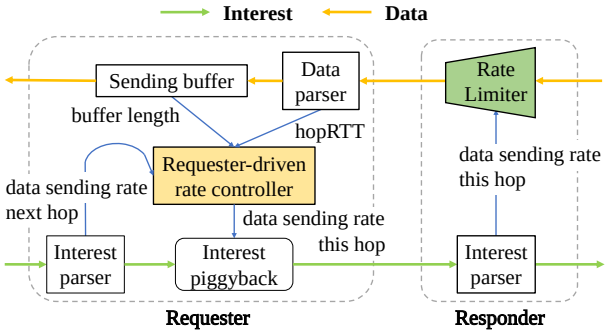


Fig. 9: Congestion control in one hop.

With the backpressure mechanism, LEOTP is able to react quickly to bandwidth variations. The reason is that congestion signals can spread from the bottleneck to the Producer at a fast speed. When the node at the bottleneck lowers its sending rate due to bandwidth variations, its buffer begins to grow as its upstream node has a higher sending rate than it. The buffer growth is fed back to the upstream node as the congestion signal. Then the upstream node lowers its sending rate too. In this way, when congestion occurs at the bottleneck, all the upstream nodes will lower their sending rate one by one. Finally, the source, i.e. Producer, will lower its sending rate, avoiding too much data being sent to the bottleneck. As congestion signals propagate back from the bottleneck directly without being carried to the Consumer first, the queuing delay and buffer overflows caused by delayed feedback are effectively reduced.

The congestion control per hop is driven by the Requester (i.e., the node sending Interest at this hop), as shown in Fig. 9. The rate controller of the Requester collects the network information and decides the sending rate of the hop. Specifically, the rate is calculated by this hop's RTT (hopRTT), the buffer length of the Requester, and the next-hop sending rate. It is piggybacked on Interests and sent to the Responder (i.e., the node responding Data at this hop). Once receiving an Interest, the Responder extracts the sending rate from it and updates its Rate Limiter, which uses this rate to control the data sending process by the token bucket algorithm.

We implement an RTT-based algorithm at each hop, which use latency variations instead of packet losses as the congestion signal. This is because packet losses can not represent the congestion well in lossy links of LEO satellite networks. Besides, we aim for low latency. So we need to adjust the sending rate when the queue starts to build up but before the loss due to buffer overflow occurs.

Like the RTT-based TCP-Vegas [20], we use $hopRTT$ to represent the actual hopRTT under load, and $hopRTT_{min}$ is the ideal value without queuing. As the responded data may not be sent immediately, the measurement of per-packet hopRTT is divided into two parts: OWD of Interest and OWD

of Data. The Requester writes a timestamp into each Interest when sending it, then the Responder calculates its OWD when receiving it. The OWD of Data is calculated in the same way. These two OWDs are added up by the Requester to get the per-packet hopRTT. The $hopRTT$ is calculated by these hopRTT samples using an exponentially weighted moving average to filter the noise. The $hopRTT_{min}$ is chosen as the minimal $hopRTT$ in the recent 5 seconds.

The congestion window *cwnd* is adjusted every hopRTT following (8), where ***BDP*** is the bandwidth-delay product of one hop and ***QueueLen*** is the estimated queuing length of the underlying layer network. They are calculated as (6) and (7). In the slow start phase and congestion avoidance phase, *cwnd* follows multiplicative increase and additive increase policy respectively. When ***QueueLen*** is greater than the threshold ***M***, LEOTP suppose congestion occurs and adjust the *cwnd* to $kBDP$, where k is chosen as 0.8 in our implementation. This is because BDP is the optimal operating point that fills up the bandwidth while avoiding congestion. We decrease *cwnd* to a value not much less than ***BDP*** to achieve faster recovery.

$$BDP = throughput * hopRTT_{min} \tag{6}$$

$$QueueLen = throughput * (hopRTT - hopRTT_{min}) \tag{7}$$

$$cwnd = \begin{cases} 2 * cwnd, & \text{if } state == SlowStart \\ cwnd + 1, & \text{else if } QueueLen \leq M \\ k * BDP, & \text{otherwise} \end{cases} \tag{8}$$

While $cwnd$ aims for high bandwidth utilization at each hop, the Requester also calculates an upper bound $rate_{bp}$ according to the backpressure algorithm for inter-hop coordination. Specifically, it is calculated by the current sending buffer length ***BL*** and next-hop sending rate $Rate_{nextHop}$ as (9), where BL_{tar} is the target length. As shown in (10), the actual sending rate ***Rate*** is limited by both the *cwnd* and the upper bound provided by the backpressure algorithm.

$$Rate_{bp} = Rate_{nextHop} + \frac{BL - BL_{tar}}{hopRTT} \tag{9}$$

$$Rate = min(\frac{cwnd}{hopRTT}, Rate_{bp}) \tag{10}$$

IV. Implementation

A. Protocol Implementation

We implement LEOTP in the user space of Linux by C++. The LEOTP packets are encapsulated in UDP transport, so there is no need for modifications to lower layers and the operating system. Our implantation is also easy to be transferred to other packet-based protocols. The passing-by UDP packets are intercepted by every intermediate node, which is transparent to endpoints. We config the netfilter [21] to achieve this goal, which redirects the packet to a local socket without changing the packet header in any way. And when the intermediate node forwards a UDP packet, it actually creates a packet whose source and destination addresses are the endpoints' addresses, which is allowed by socket option IP_TRANSPARENT [22] supported in Linux 2.6.24+. In this way, the intermediate process the packets without changing the forwarding path. So, LEOTP is compatible with any extensively studied LEO satellite network routing schemes [23], [24].

TABLE I: Packet format of LEOTP

Interest	Data	Bytes
TYPE_INT	TYPE_DATA	1
FlowID		4
rangeStart		4
rangeEnd		4
timestamp		4
sendRate	length	2
/	payload	0-MSS

The cache is implemented with HashMap and uses LRU replacement policy. The key is the ***FlowID*** and the byte range of the packet, and the value is the memory address where the payload is stored. To reduce the search time, we gather every 4096 consequent bytes in the same data flow to one block in the cache. For example, when the [100, 1500) bytes of a data flow are searched, the cache calculates that these 1400 bytes are in the block with range [0, 4096), so the cache only needs to read one block to get the data. In this way, the searching and storing of one Data packet can be finished in O(1) time.

B. Packet Format

LEOTP has two kinds of packet: Interest and Data. The total length of a LEOTP header is 15 bytes, and the Data packet has a varying size of payload. TABLE I shows their specific formats. ***FlowID*** is unique for each data flow, [***rangeStart,rangeEnd***) represents the byte-level range in a flow. These three fields form the identifier of a piece of data. The Interest uses this to indicate the data it requests, and the Data use this to present the payload it carries. *timestamp* represents when the packet is sent by the previous node, which is used to calculate ***hopRTT*** in hop-by-hop congestion control. The ***sendRate*** in Interest packets is used to inform the Responder of the expected sending rate. The *length* is consistent with the range in a normal data packet and is set as 0 in a header.

V. Evaluation

We evaluate the performance of LEOTP using the network emulator Mininet [25]. First, we present a set of experiments in controlled environments to verify the functions of LEOTP modules. Then we further compare LEOTP with the existing methods in emulated Starlink constellation.

A. Methodology

Experiment setting: The emulated Starlink experiments are based on the core constellation of Starlink [26], which has 1600 satellites evenly distributed on 32 orbital planes at an altitude of 1150km with an inclination of 53 degrees to the equator. The ground stations are supposed to be deployed in the 100 most populous cities. We support two kinds of Starlink networks. The first one is a network without ISLs,

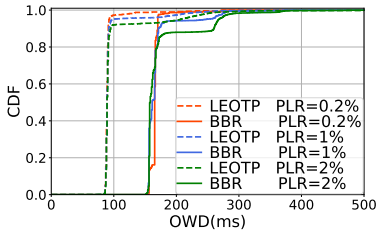


Fig. 10: The distribution of the retransmitted packets' OWD in lossy link.

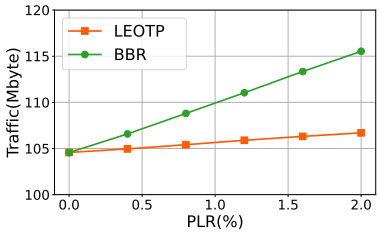


Fig. 11: The relation of loss rate and the traffic sent by sender for an 100MB file.

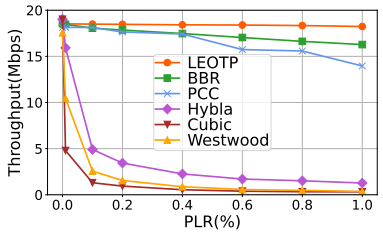


Fig. 12: The relation of loss rate and throughput.

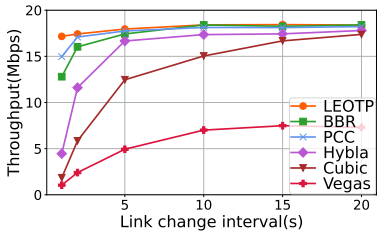


Fig. 13: The relation of topology change frequency and throughput.

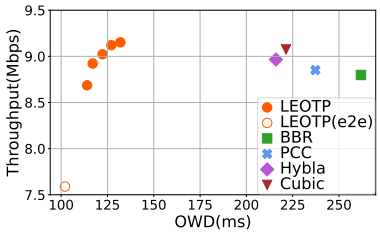


Fig. 14: Throughput-OWD trade-off under bandwidth fluctuations.

which is consistent with the current stage of Starlink. In this network, a satellite can only connect to the ground stations and the distance between the endpoints is limited. The second network has ISLs, which represents the future stage of Starlink. Transcontinental communication is enabled in this network, which is the most typical yet challenging scene in the application of LEO satellite networks [27].

Baseline: We choose several TCP variants as our baselines. In addition to Cubic [14], Hybla [15], BBR [16], and PCC [17] that appeared in the previous experiments, Westwood [28] is another loss-based variant designed for wireless links, and Vegas [20] represents the RTT-based algorithms.

B. Evaluation in Controlled Environments

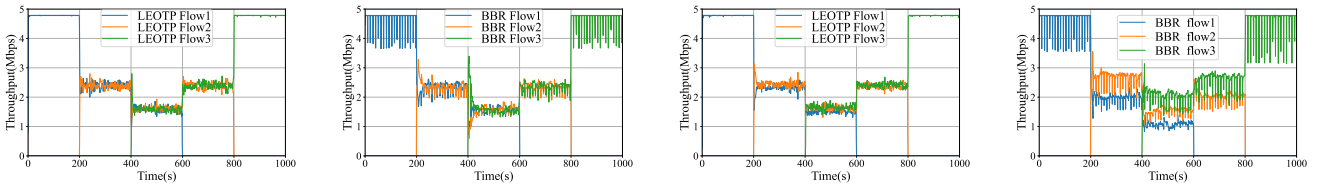
Low cost for retransmission: We first examine the function of in-network retransmission. The experiments are performed in the network with 5 hops, each of which has 20Mbps bandwidth and 20ms hopRTT. Fig. 10 shows the OWD distribution of retransmitted packets in LEOTP and the end-to-end BBR. The OWD of most retransmitted packets in BBR is about 160ms, which is an end-to-end RTT larger than the basic OWD, while the value in LEOTP is only about 90ms, for the loss can be recovered in hopRTT. With the help of in-network retransmission, LEOTP reduces 59%-64% of average recovery time under different PLR.

Fig. 11 shows the traffic the server actually sends when transmitting 100MB data in the lossy link. The traffic increases linearly with the PLR in both LEOTP and BBR, but the slope of LEOTP is about 20% of BBR. This is because all lost packets need to be resent by the end server in BBR, while in LEOTP, only 20% of packet loss happens in the first hop, which requires the server to retransmit. Packets lost at other hops can be repaired by Midnodes' cache. So LEOTP can save the precious bottleneck bandwidth. In summary, the in-network retransmission effectively reduces the retransmission delay and bandwidth waste.

High throughput against high PLR: Fig. 12 shows the throughput of LEOTP against high PLR in a 5-hop network compared with the baselines. When the per-hop PLR is 0.1%, the throughput of all the loss-based algorithms (Cubic, Hybla, Westwood) drops to less than 5Mbps. BBR and PCC also suffer a larger degradation than LEOTP, and the gap becomes more visible as the PLR rises. When the PLR per hop changes from 0 to 1%, the throughput of BBR and PCC decreases by 12% and 23%. However, LEOTP 's throughput is reduced by only 1%. The reason is LEOTP performs segmented control and each hop has better link quality than the end-to-end link.

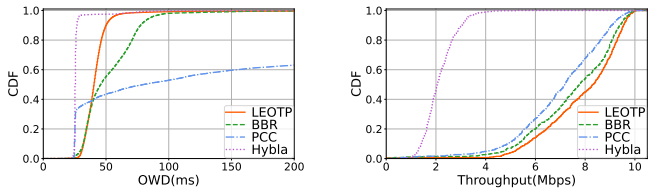
As a result, LEOTP also has better performance against link switching. In Fig. 13, we set two parallel links with different RTTs (80ms and 90ms). The bandwidth is 20Mbps in all hops. The path changes between these two links periodically. With the higher frequency of link changing, the throughput of all TCP variants decreases, as link switching causes inevitable packet loss. When the interval comes to 1s, LEOTP achieves 34% and 15% higher throughput than BBR and PCC respectively. It is also interesting to note that Vegas behaves poorly in this environment, for it is confused by the time-varying RTT. Although LEOTP uses an RTT-based congestion control algorithm in each hop, its performance is much better than Vegas. This is because, in LEO satellite networks, the end-to-end RTT fluctuates more violently due to the changing path, while the RTT of each hop is relatively stable.

Low latency under bandwidth variations: Fig. 14 shows the advantages of LEOTP under bandwidth variations. The



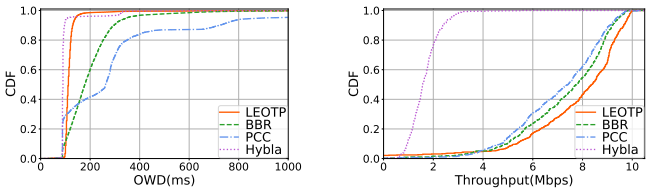
(a) LEOTP flows under same RTT (b) BBR flows under same RTT (c) LEOTP flows under different RTT (d) BBR flows under different RTT

Fig. 15: Intra-protocol fairness under same RTT and different RTT.



(a) OWD (b) throughput

Fig. 16: Cumulative distribution graph of OWD and throughput in Beijing-Shanghai link without ISLs.



(a) OWD (b) throughput

Fig. 17: Cumulative distribution graph of OWD and throughput in Beijing-New York link with ISLs.

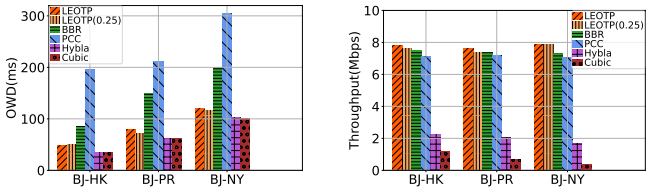
network has 10 hops, each of which has 20ms hopRTT. The second hop is set as the bottleneck with an average bandwidth of 10Mbps. It is added with a fluctuation in the form of a square wave. The period is 2s and the amplitude is 1Mbps. The bandwidth of other hops is 20Mbps. As the propagation delay is 100ms, all the TCP algorithms have a high queuing delay of over 100ms. While Cubic and Hybla fill up the buffer at the bottleneck until packet loss occurs, BBR and PCC also fail to control the queuing delay because of the long feedback cycle. Although end-to-end LEOTP achieves near-optimal latency, the throughput is quite low less than 8Mbps. This is because it can not increase its sending rate rapidly with the increase of bandwidth, and then the bandwidth drops again. However, with the assistance of intermediate nodes, LEOTP achieves both high throughput and low latency due to its quick reaction. Each point of LEOTP corresponds to a configuration of target buffer length at Midnodes. There is a trade-off between delay and throughput. A larger buffer size has a smoothing effect on bandwidth variations but also results in a longer queuing delay. In practice, we can set this parameter according to the preference of different applications.

Optimized intra-protocol fairness: To evaluate the fairness and convergence, we set up a dumbbell topology with three senders and three receivers sharing a bottleneck link with 5Mbps bandwidth and 30ms RTT, and choose BBR as the baseline. The three flows initiate sequentially with a 200s interval and run for 600s. When the three flows have the same RTT of 60ms, they converge to a similar throughput rapidly for both LEOTP and BBR, as shown in Fig. 15a and Fig. 15b. However, in LEO satellite networks, it is common that a long-distance flow and a short-distance flow share the same bottleneck. To simulate this environment, we set the three flows with 90ms, 120ms, and 150ms RTT respectively. As shown in Fig. 15c and Fig. 15d, the BBR flow with higher RTT converges to higher throughput, while LEOTP still maintains good fairness for them. This is because LEOTP performs hop-by-hop congestion control. The three flows compete in exactly the same segment, so the unfairness caused by different RTTs is alleviated. In conclusion, the fairness and convergence speed of LEOTP flows stay good in both the same-RTT and different-RTT environments.

C. Evaluation in Emulated Starlink

The emulated Starlink environment is configured as follows: (i) We use the parameters of the Starlink constellation. We calculate the satellite locations and the routing result at any time by the route computing module of HYPATIA [29], which uses the Floyd-Warshall algorithm. The hopRTT is calculated by the distance and the speed of light. (ii) GSL uplink is the bottleneck with a maximum bandwidth of 10Mbps. The bandwidth of other hops is 20Mbps. (iii) The PLR of GSLs and ISLs is 1% and 0.1% respectively. (iv) The bandwidth of GSL changes before and after handover like the "V" curve based on real trace [30]. A random bias within $\pm$0.5Mbps is added to simulate bandwidth fluctuations.

In the LEO satellite network without ISLs, we evaluate the performance of the Beijing-Shanghai link, which is shown in Fig. 16. Hybla's throughput is far below the available bandwidth because of packet loss. Hence no congestion occurs and its delay is near-optimal. BBR and PCC achieve much higher throughput, but their queuing delay is obvious under bandwidth variations. As shown in Fig. 16a, while BBR has an acceptable average queuing delay of 26ms, PCC's queuing delay is more than 400ms. With the advantages of hop-by-hop congestion control, the throughput of LEOTP is 4.8% higher than BBR, and 12.4% higher than PCC. It also reduces the average queuing delay to 16ms, which is 0.61x of BBR.



(a) OWD (b) throughput

Fig. 18: Average OWD and throughput in different city pair links with ISLs.

In the LEO satellite network with ISLs, we evaluate the links from Beijing to Hong Kong, Paris, and New York. The distance from Beijing to these cities is 1968km, 8209km, and 10991km, and the average hop count is 5.15, 9.53, and 18.88 respectively in our emulation. Fig. 17 shows the result of the Beijing-New York link. In Fig. 17b, LEOTP achieves about 8.0% higher throughput compared to BBR and 12.2% to PCC, which is consistent with the environment without ISLs. However, Fig. 17a shows that BBR also suffers a significantly high queuing delay of 100ms due to the increased feedback loop, while LEOTP maintains a low average queuing delay of only 20ms. The effect of in-network retransmission is also obvious with a large hop count and high end-to-end PLR. The 99th percentile OWD of LEOTP is even 53ms lower than Hybla, as the retransmission delay is significantly shortened.

In Fig. 18, we compare the results on the three links to show how distance affects the performance of LEOTP and TCP variants. As shown in Fig. 18a, with the increase of the distance, the OWD of BBR and PCC increase rapidly because of the problem of delayed feedback, while LEOTP always keeps its OWD only 15ms-20ms higher than the ideal propagation delay. Fig. 18b shows that when the hop count increases, the throughput of Cubic and Hybla decreases with higher PLR. The throughput of LEOTP has no visible degradation and maintains at least 4% higher than BBR and PCC. So we conclude that **LEOTP has more significant improvement in long-distance communications**. Interestingly, we also note that **LEOTP can achieve good performance with the assistance of a small amount of LEO satellites**. With only 25% coverage, LEOTP achieves higher throughput than BBR and PCC in all environments, with a delay slightly lower than that of full coverage. This means LEOTP's Midnodes can be flexibly deployed on part of the satellites to save costs.

At last, we show the respective contributions of the key modules in LEOTP through an ablation experiment. The result is shown in TABLE II, where A is LEOTP with complete functions, B enables hop-by-hop congestion control but has no cache, C enables in-network retransmission but the congestion control is performed by endpoints, and D has no Midnodes. When comparing A to C, and B to D respectively, we can see hop-by-hop congestion control increases throughput significantly in all scenarios, as it has a much quicker reaction to changes of network conditions. In contrast, the end-to-end congestion control of LEOTP is too conservative to achieve high link utilization. And when comparing A to B, and C to D, we can see the effect of in-network retransmission in both reducing latency and increasing throughput. The enhancement becomes more apparent with the increase of distance and PLR. In conclusion, **both of the two modules contribute to the better performance of LEOTP in LEO satellite networks.**

TABLE II: The result of the ablation experiment

	BJ-HK		BJ-PR		BJ-NY	
	Throughput (Mbps)	**OWD (ms)**	**Throughput (Mbps)**	**OWD (ms)**	**Throughput (Mbps)**	**OWD (ms)**
A	7.82	49.17	7.70	76.57	7.91	118.64
B	7.78	51.39	7.67	80.74	7.73	126.10
C	7.38	40.15	7.23	66.40	6.80	103.63
D	7.24	42.05	7.03	70.38	6.52	112.20

VI. Related Work

End-to-end transmission control. The classic TCP variants designed for satellite networks such as Hybla and Peach [31] aim at GEO satellite links, which is of little help in LEO satellite networks. In recent years, several congestion control algorithms for LEO is proposed. CCOSPF is based on Open Short Path First (OSPF) [32]. [33] modifies the congestion avoidance algorithm based on Westwood. Meanwhile, [12] indicates QUIC can outperform TCP in space network scenarios due to its zero handshaking and loss recovery mechanism. However, these methods have the general limitations of end-to-end transmission and lack of concern for queuing delay.

Proxy. Split TCP is the typical form of performance-enhancing proxy (PEP). Ack filtering [34] uses proxies to filter out part of the ACKs, which improves TCP's performance for highly asymmetric links. Snoop proxy [35] caches packets for local retransmission and hides packet loss from the TCP sender. However, the proxy does not perform loss detection and the local retransmission only happens on the last hop.

ICN. ICN architecture uses a name-based rather than a host-centric communication model. The transmission is pull-based and driven by the receiver, and point-to-multipoint communication is supported by in-network caching. Recent studies [36], [37] investigate the ICN's architectural benefits in LEO satellite networks, such as mobility support and adaptive forwarding. However, the current applications and protocols are not compatible with ICN, so deploying ICN to LEO satellite networks directly is difficult. INTCP [38] leverages the information-centric paradigm in the transport layer for data transmission in dynamic topology, but it lacks further design on reliable transfer and congestion control.

Link layer control. FEC (Forward Error Correcting) and link-level retransmission are designed to reduce BER on wireless links. However, a large portion of packet loss is caused by congestion [5] and link switching [13] in LEO satellite networks, which can not be repaired in link layer. In addition, the independent retransmission in link layer and transport layer may interfere with each other and lead to severe performance degradation [39]. BFC [40] implements per-hop flow control in the link layer, but this requires hardware support on each switch, which is costly for LEO satellite networks.

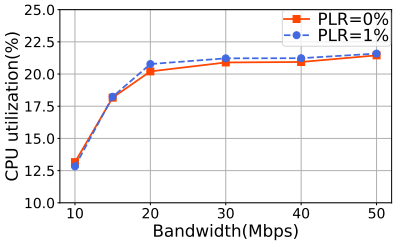


Fig. 19: The CPU overhead of LEOTP.

VII. Discussion

Scalability. Starlink claims that today's LEO constellations have a decent amount of general operating systems and programming environments [41], which provides possibilities to run LEOTP. Meanwhile, the overhead of LEOTP is low. First, the algorithms have low complexity. So a Midnode spends most of its processing time on I/O of packets. In Fig. 19, we measure the CPU utilization of a Midnode under different bandwidths and PLR. It shows that LEOTP maintains a low CPU utilization. When the bandwidth exceeds 20Mbps, it grows with bandwidth at a quite low speed and is not sensitive to packet loss. Second, the cache at Midnodes mainly works for in-network retransmission, so it only needs to store the packets received in the past hopRTT. The hopRTT is usually a few milliseconds, and a satellite can only communicate with 4 other satellites at the same time [27]. So a cache of a few hundred MB is enough even if the link capacity of ISLs reaches tens of GB, which is trivial compared to the memory of modern devices. Moreover, the optimization of caching strategy can further save memory usage. Third, although each Midenode has to keep some per-flow states (e.g., congestion status and sequence numbers), they are only tens of bytes for each flow. Besides, these states are short-term active so can be reconstructed rapidly upon failures.

Multicast. With the information-centric model, LEOTP has the potential of multicast inherently. When several Consumers request the same data at the same time, the cache in Midnodes could block the duplicate Interests and respond data immediately, avoiding duplicate data transmitting from upstream nodes. This can be done if the Consumers share the same *FlowID*. However, the multicast function puts higher requirements on the cache. If the cache can only store a piece of data for several milliseconds, it is hard for other flows to fetch this data. While the satellites are fast-moving and have limited capabilities, a hierarchical cache at ground nodes that leverages the disk can be designed to solve this problem.

Compatible with TCP. An alternative solution is to use LEOTP only in the satellite segment. Transparent proxies are deployed at ground stations to connect the territorial network and LEOTP. In this way, the enhancement can be achieved without change at the endpoints, which is more practical. However, TCP is sender-driven with a stateful connection, while LEOTP is a connectionless receiver-driven protocol, so the bridging between them can be hard. The gateway designed for carrying TCP/IP traffic in ICN networks [42], [43] may be helpful to solve the contradiction problem.

VIII. Conclusion

To provide high-speed and low-latency Internet access on LEO satellite links, it is necessary to make great innovations on the transport layer. To this end, we propose an information-centric transport layer protocol LEOTP. The Request-Response model with in-network caching is the basis of the transport layer, which supports the mobility of the intermediate satellite nodes. The reliability is achieved at a minimal cost of delay and bandwidth consumption with the help of in-network retransmission. The hop-by-hop congestion mechanism provides accurate traffic control, avoiding congestion in the network while maximizing link utilization. The extensive experiments validate that LEOTP improves both throughput and delay in LEO satellite networks remarkably.

Acknowledgements

We thank the anonymous reviewers for their valuable feedbacks. This work was sponsored by the NSFC grant(U21B2012, 62032013, 62272258) and Guangdong Basic and Applied Basic Research Foundation(2019B1515120031). We gratefully acknowledge the support of State Key Laboratory of Media Convergence Production Technology and Systems and Key Laboratory of Intelligent Press Media Technology. Xinggong Zhang is the corresponding author.

References

[1] Jonathan's Space Report. Starlink statistics, November 2022. https://planet4589.org/space/stats/star/starstats.html.

[2] Yvon Henri. The oneweb satellite system. *Handbook of Small Satellites: Technology, Design, Manufacture, Applications, Economics and Regulation*, pages 1–10, 2020.

[3] Alan Boyle. Amazon to offer broadband access from orbit with 3,236-satellite 'Project Kuiper' constellation, April 2019. https://www.geekwire.com/2019/amazon-project-kuiper-broadband-satellite/.

[4] Inigo Del Portillo, Bruce G Cameron, and Edward F Crawley. A technical comparison of three low earth orbit satellite constellation systems to provide global broadband. *Acta astronautica*, 159:123–135, 2019.

[5] François Michel, Martino Trevisan, Danilo Giordano, and Olivier Bonaventure. A first look at starlink performance. In *Proceedings of the 22nd ACM Internet Measurement Conference*, IMC '22, page 130–136, New York, NY, USA, 2022. Association for Computing Machinery.

[6] Debopam Bhattacherjee and Ankit Singla. Network topology design at 27,000 km/hour. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, CoNEXT '19, page 341–354, New York, NY, USA, 2019. Association for Computing Machinery.

[7] Deepak Vasisht, Jayanth Shenoy, and Ranveer Chandra. L2d2: Low latency distributed downlink for leo satellites. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 151–164, New York, NY, USA, 2021. Association for Computing Machinery.

[8] Swastik Kopparty, Srikanth V Krishnamurthy, Michalis Faloutsos, and Satish K Tripathi. Split tcp for mobile ad hoc networks. In *Global Telecommunications Conference, 2002. GLOBECOM'02. IEEE*, volume 1, pages 138–142. IEEE, 2002.

[9] Stepanov Evgeniy and Voynov Nikita. Dynamic split tcp. In *2020 International Scientific and Technical Conference Modern Computer Network Technologies (MoNeTeC)*, pages 1–9. IEEE, 2020.

[10] Zouhair El-Bazzal, Abdel Mehsen Ahmad, Mohamad Houssini, Ibrahim El Bitar, and Zainab Rahal. Improving the performance of tcp over wireless networks. In *2018 Sixth International Conference on Digital Information, Networking, and Wireless Communications (DINWC)*, pages 12–17. IEEE, 2018.

[11] Li Jiang, Yihang Zhang, Jinyu Yin, Xinggong Zhang, and Bin Liu. Leotp source code, April 2023. https://jl99888.github.io/LEOTP.

[12] Siyu Yang, Hewu Li, and Qian Wu. Performance analysis of quic protocol in integrated satellites and terrestrial networks. In *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pages 1425–1430, 2018.

[13] Jirui Zhang, Shibing Zhu, Hefeng Bai, and Changqing Li. Optimization strategy to solve transmission interruption caused by satellite-ground link switching. *IEEE Access*, 8:32975–32988, 2020.

[14] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS operating systems review*, 42(5):64–74, 2008.

[15] Carlo Caini and Rosario Firrincieli. Tcp hybla: a tcp enhancement for heterogeneous networks. *International journal of satellite communications and networking*, 22(5):547–566, 2004.

[16] Neal Cardwell, Yuchung Cheng, S Hassas Yeganeh, Ian Swett, Victor Vasiliev, Priyaranjan Jha, Yousuk Seung, Matt Mathis, and Van Jacobson. Bbrv2: A model-based congestion control. In *Presentation in ICCRG at IETF 104th meeting*, 2019.

[17] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, P. Brighten Godfrey, and Michael Schapira. Pcc vivace: Online-learning congestion control. In *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation*, NSDI'18, page 343–356, USA, 2018. USENIX Association.

[18] Yannick Hauri, Debopam Bhattacherjee, Manuel Grossmann, and Ankit Singla. "internet from space" without inter-satellite links. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, HotNets '20, page 205–211, New York, NY, USA, 2020. Association for Computing Machinery.

[19] Matt Sargent, Jerry Chu, Dr. Vern Paxson, and Mark Allman. Computing TCP's Retransmission Timer. RFC 6298, June 2011.

[20] L.S. Brakmo and L.L. Peterson. Tcp vegas: end to end congestion avoidance on a global internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, 1995.

[21] The Netfilter's webmasters. netfilter/iptables project homepage - The netfilter.org project, January 2021. https://www.netfilter.org/.

[22] Jambit GmbH. ip(7) — linux manual page, February 2021. https://man7.org/linux/man-pages/man7/ip.7.html.

[23] Tian Pan, Tao Huang, Xingchen Li, Yujie Chen, Wenhao Xue, and Yunjie Liu. Opspf: orbit prediction shortest path first routing for resilient leo satellite networks. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.

[24] WANG Xuan, HOU Rong-hui, and XU Wei-lin. Dynamic path switching technology for leo satellite networks. *Journal of Beijing University of Posts and Telecommunications*, 43(2):80, 2020.

[25] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pages 1–6, 2010.

[26] Jonathan C McDowell. The low earth orbit satellite population and impacts of the spacex starlink constellation. *The Astrophysical Journal Letters*, 892(2):L36, 2020.

[27] Mark Handley. Delay is not an option: Low latency routing in space. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, pages 85–91, 2018.

[28] Saverio Mascolo, Claudio Casetti, Mario Gerla, M. Y. Sanadidi, and Ren Wang. Tcp westwood: Bandwidth estimation for enhanced transport over wireless links. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, MobiCom '01, page 287–297, New York, NY, USA, 2001. Association for Computing Machinery.

[29] Simon Kassing, Debopam Bhattacherjee, André Baptista Águas, Jens Eirik Saethre, and Ankit Singla. Exploring the "internet from space" with hypatia. In *Proceedings of the ACM Internet Measurement Conference*, IMC '20, page 214–229, New York, NY, USA, 2020. Association for Computing Machinery.

[30] Kiruthika Devaraj, Matt Ligon, Eric Blossom, Joseph Breu, Bryan Klofas, Kyle Colton, and Ryan W Kingsbury. Planet high speed radio: Crossing gbps from a 3u cubesat. 2019.

[31] Ian F Akyildiz, Giacomo Morabito, and Sergio Palazzo. Tcp-peach: a new congestion control scheme for satellite ip networks. *IEEE/ACM Transactions on networking*, 9(3):307–321, 2001.

[32] Siyuan Cao and Tao Zhang. Congestion control based on ospf in leo satellite constellation. In *2019 IEEE 19th International Conference on Communication Technology (ICCT)*, pages 1111–1115. IEEE, 2019.

[33] Fahrul Hakim Ayob, Shamala Subramaniam, Mohamed Othman, and Zuriati Zulkarnain. An enhanced congestion control algorithm for leo satellite networks. *International Journal of Engineering & Technology*, 7(4.31):363–367, 2018.

[34] Yoshiaki Ohta, Michiharu Nakamura, Yoshihiro Kawasaki, and Takayoshi Ode. Controlling tcp ack transmission for throughput improvement in lte-advanced pro. In *2016 IEEE Conference on Standards for Communications and Networking (CSCN)*, pages 1–6. IEEE, 2016.

[35] Milosh V Ivanovich. 5 enhancing tcp performance in hybrid networks. *Internet Networks: Wired, Wireless, and Optical Technologies*, page 99, 2018.

[36] Zhongda Xia, Yu Zhang, Teng Liang, Xinggong Zhang, and Binxing Fang. *Adapting Named Data Networking (NDN) for Better Consumer Mobility Support in LEO Satellite Networks*, page 207–216. Association for Computing Machinery, New York, NY, USA, 2021.

[37] Teng Liang, Zhongda Xia, Guoming Tang, Yu Zhang, and Beichuan Zhang. Ndn in large leo satellite constellations: A case of consumer mobility support. In *Proceedings of the 8th ACM Conference on Information-Centric Networking*, ICN '21, page 1–12, New York, NY, USA, 2021. Association for Computing Machinery.

[38] Jinyu Yin, Li Jiang, Xinggong Zhang, and Bin Liu. Intcp: Information-centric tcp for satellite network. In *2021 4th International Conference on Hot Information-Centric Networking (HotICN)*, pages 86–91, 2021.

[39] A. DeSimone, Mooi Choo Chuah, and On-Ching Yue. Throughput performance of transport-layer protocols over wireless lans. In *Proceedings of GLOBECOM '93. IEEE Global Telecommunications Conference*, pages 542–549 vol.1, 1993.

[40] Prateesh Goyal, Preey Shah, Kevin Zhao, Georgios Nikolaidis, Mohammad Alizadeh, and Thomas E. Anderson. Backpressure flow control. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 779–805, Renton, WA, April 2022. USENIX Association.

[41] Liam Tung. Spacex: We've launched 32,000 linux computers into space for starlink internet, June 2020. https://www.zdnet.com/article/spacex-weve-launched-32000-linux-computers-into-space-for-starlink-internet/.

[42] Ran Zhu, Tianlong Li, and Tian Song. igate: Ndn gateway for tunneling over ip world. In *2021 International Conference on Computer Communications and Networks (ICCCN)*, pages 1–9, 2021.

[43] Ilya Moiseenko and Dave Oran. Tcp/icn: Carrying tcp over content centric and named data networks. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, ACM-ICN '16, page 112–121, New York, NY, USA, 2016. Association for Computing Machinery.