To Punctuality and Beyond: Meeting Application Deadlines with DTP

Jie Zhang*, Hang Shi†, Yong Cui*, Feng Qian‡, Wei Wang†, Kai Zheng†, Jianping Wu*§
Tsinghua University, Huawei Technologies†, University of Minnesota - Twin Cities‡, Zhongguancun Laboratory§*

***Abstract*—Many applications have deadline requirements for their data delivery, such as real-time video, multiplayer gaming, and cloud AR/VR. However, the current transport layers' APIs are too primitive to accomplish that. Therefore, today's applications are forced to build their customized and complex deadline-aware data delivery mechanisms. In this work, we design Deadline-aware Transport Protocol (DTP) to provide deliver-before-deadline service over the wild Internet. To fulfill the diverse and sometimes conflicting requirements over the fluctuating network, we design the Active-Drop-at-Sender scheduler and adaptive redundancy. We build DTP by extending QUIC, and then develop two applications that utilize DTP. Extensive evaluations demonstrate that DTP is easy to use and can bring significant performance improvement (1.2x to 5x) compared to vanilla QUIC.**

I. Introduction

Many applications have requirements for their data to arrive before a certain time, *i.e.,* deadline. Data missing the deadline is useless to the application because it will be obsoleted by newer data. Such applications include real-time media and online multiplayer gaming. For example, the end-to-end delay of a video conferencing system should be below a tolerable threshold (about 100ms) to enable participants to interact naturally [1]. Novel applications such as mobile virtual reality (VR) offloading require the motion-to-photon latency below 25ms to avoid motion sickness [2]. For online multiplayer gaming, the server aggregates each player's actions every 60ms and distributes those to other players, so each player's state is kept in sync with the server. Packets missing this round of synchronization will be overwritten by future packets. In other words, they all have a deadline requirement for their transmission.

A large body of work in several communities aims at meeting deadline requirements for data delivery. Layer 2 and 3 techniques [3], [4] and in-network cooperation effort [5], [6] require either modification to in-network hardware or cooperation between end hosts and in-network elements, which is not feasible on the wild Internet. Traditional coupled solutions [7], [8] and extension to transport protocols [9]–[12] all suffer from deployment issues. In this work, we instead focus on providing the deadline-aware delivery service for applications over the wild Internet. Designing such a transport protocol is challenging on multiple fronts.

1) Applications' transmission requirements are highly diverse. The protocol should provide a general abstraction for applications to express their requirements and provide useful metadata.
2) Unlike the data center network, we can neither change the in-network hardware nor regulate other competing connections' behaviors, so the delay inside the network is out-of-control.
3) Each block may have distinct and conflicting properties such as deadline, priority, and dependency.

To address the above challenges, we first analyze the common requirements and properties of representative emerging applications (Section II) and find that they all have multiple data blocks of different priorities, making it possible to allocate network resources to high priority ones. Based on those insights we propose a generic and easy-to-use block-based protocol, which we call DTP[1], for deadline-aware data transfer. We then analyze the latency components of the delivery process and find that we can reduce the delay at the sender and retransmission time by Active-Drop-at-Sender (ADS) scheduling and using redundancy respectively (Section III). Our ADS scheduler strategically balances multiple conflicting factors including deadline, priority, and dependency when deciding which block to send or drop. The redundancy module is only activated when the deadline is about to miss, thus avoiding retransmission delay without incurring too much overhead.

Using DTP, applications can convey the deadline requirements together with necessary metadata and get meaningful feedback. DTP tries to fulfill needs of applications and notify applications of delivery results. Moreover, DTP can carry delivery results from the receiver to the sender and let the sender adjust the requirements. DTP can thus significantly reduce the burden on application developers.

QUIC [13], [14] is a promising protocol that provides more precise RTT and bandwidth estimations, as well as loss detection. QUIC prevents most of the interferences from middleboxes thus more deployable [14]. These may shed new light on deadline-aware delivery improvement and deployment. We implement DTP based on QUIC in a non-intrusive way (Section IV) as full user-space system. The evaluation results show that DTP can deliver 5x blocks before the deadline than QUIC when the bandwidth is limited, and 1.2x blocks when there are packet losses. We develop a 360° video streaming application. The evaluation indicates that using DTP can improve the QoE by 30% to 500%. We build a multi-party video conferencing application using FFMPEG and TCP. Then we only modify

978-1-6654-8234-9/22/$31.00 ©2022 IEEE

[1]Prototype of DTP is available at https://github.com/STAR-Tsinghua/DTP

98 lines of code out of 5K LoC to integrate DTP and get 30% improvement on the QoE (Sections V and VI).

To summarize, our contributions are listed as follows:

1) We design DTP, a new transport protocol with deadline-awareness support. At the sender, the application sends the data and its metadata to DTP. DTP uses the properties of blocks to schedule block transmissions. Meanwhile, DTP leverages redundancy to avoid retransmission when the deadline is about to miss.
2) We implement DTP as an extension to QUIC. The evaluation shows that the performance is 1.2x to 5x of QUIC over a wide range of network conditions.
3) We develop two applications using DTP. Evaluation shows DTP can improve their QoE up to 5x.

II. Background and Motivation

Emerging applications such as 360° video and cloud VR/gaming have the deadline requirement for their data transfer. We first conduct case studies of these applications and then motivate our new transport protocol.

A. Case studies

360° video streaming. Many researchers developed tile-based streaming [15], [16] for 360° video. The panoramic scene is divided into many tiles [17], [18] which get transmitted, and decoded in parallel. The delay of any tile in the viewport will stall the whole scene. Therefore, each tile has a deadline for its transmission (= playtime - local decode time). To reduce bandwidth consumption, only important tiles (based on the user's attention) are transmitted. Cross-tile encoding will introduce a dependency between those tiles.

Cloud VR gaming. Many studies try to leverage the cloud or edge computation power to render high-quality VR content and stream it back to the mobile device. Lai *et al.* [2], Kämäräinen *et al.* [19] develop a cooperative rendering architecture. The mobile device fetches and renders the dynamic foreground object while the cloud renders the background scene, and then streams it as a panoramic scene. Liu *et al.* [20] use the display VSync signal to synchronize the client displaying and remote server rendering. Missing the VSync signal can lead to severe display delay. This VSync signal can be viewed as the deadline of the object transmission.

Cooperative augmented vehicular reality. Autonomous vehicles use a variety of sensors such as LiDAR, Radar and stereo cameras to sense their surroundings. However, they have limited range and only have line-of-sight visibility, so they cannot perceive far away or occluded objects. To overcome those limitations, Qiu *et al.* [21] proposed the Augmented Vehicular Reality (AVR), which extends the visual horizon of vehicles by sharing visual information between each other wirelessly. To reduce the bandwidth requirement of communication, the sender can prioritize some objects over others based on their distance/velocity.

B. Insights

We summarize the common transport requirements of those applications as follows:

Deadline requirement for block-based data transmission. Those applications all generate and process the data in a block fashion and each block has a clear deadline requirement. The block is defined as the *minimal usable unit* of data for applications. A partially delivered block is useless for those applications. For example, video/audio encoders produce the encoded streams as a series of blocks (I, B, P frame, or GOP). In multiplayer games, the player's commands and world state will be bundled as a message. For web browsing, HTML, CSS, and JS can be treated as objects. Even for file syncing, most file cloud system syncs the data on chunk bases.

The meaningful deadline for an application is the block completion time, *i.e.,* the time between when the block is generated at the sender and submitted to the application at the receiver. If a block cannot arrive before the deadline, the QoE will be affected and the whole block is useless (obsoleted by newer blocks or no longer needed). In 360° video, each tile in one GOP is a block and the deadline of tiles and objects is its render time minus the local processing time. If the deadline is missed, the video will stall or go black. In cloud VR gaming, each background scene and foreground object is a block. A late arrival object is useless because the user's interest changes. In autonomous driving, each 3D object is a block and its deadline is set to the time when it is no longer occluded.

Multiple concurrent blocks with different properties. There are usually many blocks that need to be transferred in parallel. These blocks have different impacts on application performance, making it possible to reduce bandwidth consumption by prioritizing some blocks over others.

In 360° video, the priority of the tile can be set according to the user's viewport. The tile in the center of the viewport has a higher priority compared to peripheral tiles.

In cloud VR gaming, foreground objects have a higher priority and tighter deadline than background scenes. Different foreground objects can also have different priorities and deadlines based on the possibility of user interaction.

In autonomous driving, different dynamic objects have different priorities. One way to set the priority of an object is to set it by how dangerous it is to other vehicles. For example, the fast oncoming vehicle has a higher priority over the following vehicle than a slow-moving and far-away bicycle.

Block dependency. Applications may use complicated encoding to compress the block data, which usually involves cross-block delta compression. A typical example is I/P frames in H264 and base/enhance layers in the SVC codec. Those cross-block compressions can significantly reduce the bandwidth requirement but introduce dependencies between blocks. If the depended block can not arrive in time, then there is no point for the depending block to arrive. Therefore, the dependency should be respected when sending blocks.

Application level adaptation. To deal with network fluctuation, the applications usually develop their adaptation logic. For example, they usually try to match the data sending rate to the monitored/predicted bandwidth. 360° video streaming uses an ABR algorithm to pick the video quality. Cloud VR gaming reduces the frame rate or resolution of the background scene. How to adjust the behavior is highly specific to each type of application. The adaptation may happen at the sender or receiver. However, they all need network status feedback from the transport layer, at least the delivery results of blocks.

C. Limitations of current transport protocols

TABLE I: Comparison of different transport protocols

Features	TCP	UDP	SCTP	QUIC	DTP
Block-based	No	No	No	No	**Yes**
Deadline-aware	No	No	No	No	**Yes**
Multiplex	No	No	Yes	Yes	**Yes**
Dependency	No	No	No	No	**Yes**
Priority	No	No	No	Yes	**Yes**

In short, those applications need a transport protocol that can provide a multi-block based deliver-before-deadline service. It should support multiplex, *i.e.,* transferring multiple blocks in parallel. Each block needs to have its own property such as deadline, priority, and dependency. No existing transport protocol can provide such a service (Table I).

Without proper transport protocol support, applications are forced to build their own wheels. A typical example is the WebRTC which incorporates everything from encoding to transport, resulting in millions lines of code. Instead, those applications often opt to use off-the-shelf protocols such as TCP which may result in inferior performance. Moreover, some wheels are not suitable to be mounted in the application. For example, although the application can reduce the delay caused by packet loss using redundancy, doing so will be complex or ineffective because it needs information about packets sent, loss detection and other transport layer features. Ideally, the application only cares about what to send and the transport layer determines how to send. We propose Deadline-aware Transport Protocol (DTP) to provide such a service. DTP uses the Active-Drop-at-Sender scheduler to address delay at the sender, and the adaptive redundancy to relieve the delay caused by loss recovery.

III. Design

The design of DTP follows three principles: 1) End-to-end principle so that DTP does not require the modification of in-network devices. DTP should place the data at the endpoint which it can control as much as possible. 2) Adaptive to diverse application requirements such as deadline, priority, and dependency. 3) Reduce the burden on application developers. DTP handles the short-term fluctuation and the application only needs to select the appropriate setting according to the long-term QoE or data delivery results.

To accomplish the deliver-before-deadline task, we first need to understand where the latency comes from. We can divide the block delivery time into three parts:

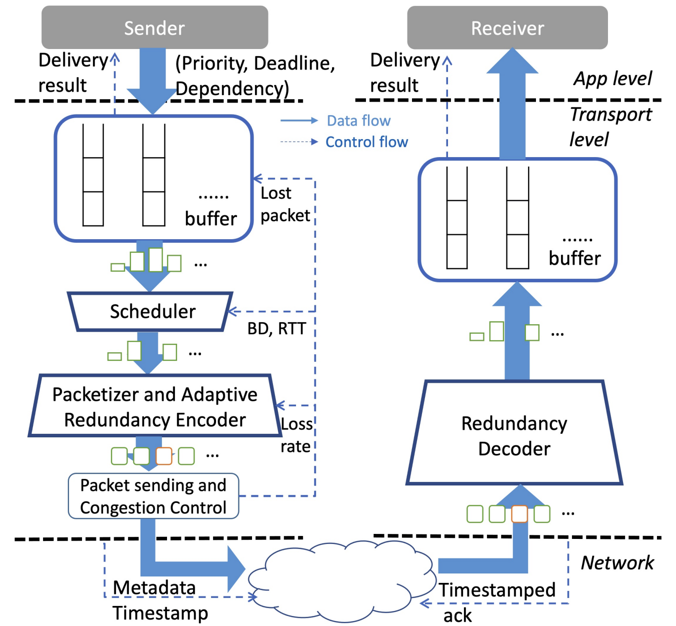


Fig. 1: The architecture of DTP

1) Delay at the sender. When the application's data rate is higher than the bottleneck bandwidth, the data will be buffered. This delay will affect all later blocks.
2) Delay in the network. This includes the data transmission delay($\frac{Blocksize}{bandwidth}$) and RTT.
3) Delay at the receiver. The receiver needs to wait for the lost or reordered packets to arrive. Loss detection and retransmission also take time. They can not be ignored especially when the tail packets of the block lost.

The delay in the network is determined by how many network resources the connection can get. A TCP-friendly transport can not get more bandwidth than its fair share. The RTT is affected by the amount of occupied network buffer. Buffer-filled congestion control may get higher bandwidth but also higher RTT and possibly higher loss.

We use an Active-Drop-at-Sender Scheduler (Section III-B) to reduce the sender queue, thus reducing the delay at sender, especially for important blocks. To get low RTT and keep schedulable data in the sender as much as possible, DTP needs to employ congestion controls that do not aggressively fill the network buffer. When the link is lossy and the deadline is tight, the loss of tail packets will cause the block to miss its deadline. The redundancy module Section III-C can mitigate that problem by recovering lost packets without waiting for loss detection and retransmission.

A. Architecture

The architecture of DTP is shown in Fig. 1, in which the transport service modules are organized in a pipeline. DTP uses standard low-latency congestion control algorithm (BBR v2 [22]) to get the lowest RTT. By using BBR, DTP shifts the queue from the network to the sender where DTP can control. Other congestion control algorithms which do not result in much in-network queue, such as Vegas [23], are also suitable.

DTP does not customize the congestion control thus inducing no bandwidth sharing issues such as fairness problems.

Each *block* along with its metadata is firstly stored in a dedicated sending queue. Then the *Scheduler* module will select the block to send and drop stale blocks. The *Packetizer and Redundancy* module will break the block into a packet stream and generate redundant packets to reduce the retransmission delay if necessary. These packets will be sent to the *packet sending and congestion control* module which sends packets, collect ACK and detect loss, similar to QUIC/TCP. The lost packet is put in front of the respective sending queue thus prioritized naturally. The congestion control module monitors the network condition and provides estimation of bandwidth, RTT and loss rate to the scheduler and redundancy module, facilitating the scheduling and redundancy decision. On the receiver, the transport layer will receive data and reassemble blocks. The process is symmetrical to the sender.

To calculate the block completion time, the sender and the receiver need to sync some information with each other. The receiver needs to know the block begin time (when the block arrives at the DTP). The sender needs to know the block end time (when all packets of a block are received or restored by receiver). We choose to send the block begin time to the receiver and send back the packet receive time back along with the ACK to the sender (see Section IV-B). In this way, both sides can know the block completion time. Sender can estimate the one way delay using the timestamp from ACK. We also design mechanisms to handle the unsynchronized clock between endpoints (see Section IV-D).

B. Active-drop-at-sender (ADS) scheduler

As discussed in the previous section, the scheduler is responsible for manipulating the sender side queue to reduce the queuing delay for those important blocks. The goal of the scheduler is to deliver as many high priority blocks as possible before the deadline, dropping those unimportant blocks if necessary. Meeting the goal is challenging because there are multiple conflicting factors involved when making the scheduler decision. The first is the conflict between higher-priority blocks far from deadlines versus lower-priority blocks with close deadlines. Sending the former may cause the latter to miss the deadline, while sending the latter may hinder the delivery of the former. The second is the conflict between blocks in transmission versus more important blocks waiting to be transmitted. Preemptively transmitting the latter means dropping the former, causing waste of network resources, while waiting for the former to finish may cause the latter to miss their deadlines.

Regarding the multiple factors mentioned above, a naive algorithm will only consider the priority when picking the block to send. However, this simple algorithm may not get optimal results. Suppose the bandwidth reduces and the scheduler chooses not to send the low priority block. Then the bandwidth is restored. The data block with lower priority is closer to the deadline than the high priority block. If in this round the scheduler still chooses to send the high priority block, then the low priority block may miss the deadline next round and become useless. In some cases, the scheduler can choose to send a low priority block because it is more urgent. However, it should do so without causing the high priority blocks to miss their deadlines. This example reveals a fundamental conflict between the application specified priority and deadline implicated priority. We need to take both types of priorities into consideration when scheduling blocks.

Another decision the scheduler should make is when to interrupt the transmission of a block and switch to a new one, *i.e.,* preemption. A non-preemptive scheduler is clearly not optimal. Consider this simple case. While a big block is already in the transmission, a higher priority tight-deadline block comes and needs to be sent immediately; otherwise it will miss the deadline. If the scheduler is not preemptive, the new block will not get sent while the bandwidth is wasted to transfer the older, less-important block. Therefore, the scheduler must be preemptive. But when to perform a preemptive transfer? If the scheduler preempts too aggressively, blocks will be switched in and out too frequently and none of them may finish before the deadline. DTP uses the block unsent ratio (how much data of a block has not been already sent yet) to determine if the block should be switched out.

$$remain_time = deadline - \frac{remain_size}{estimated_bandwidth} - passed_time - one_way_delay \quad (1)$$

$$f(remain_time) = \begin{cases} \frac{remain_time}{deadline}, remain_time > 0 \\ min(\frac{-remain_time}{deadline}, 1) + \beta, otherwise \end{cases} \quad (2)$$

$$Weighted_p = ((1-\alpha) \times f(remain_time) + \alpha \times \frac{priority}{p_{max}}) \times unsent_ratio \quad (3)$$

ADS scheduler uses Eq. (3) to compute a $Weighted_p$ (smaller value means higher priority) from the following factors:

1) Remaining time to the deadline (Eq. (1)). Once a block is deemed overdue, it will get a bigger $Weighted_p$ value by adding β (Eq. (2)). The bigger β is, the more strict the deadline is. When the β is large enough, blocks that are judged to be missed will only be considered if there are no fresh blocks to send.
2) Priority. The priority in Eq. (3) considers both the application specified priority and deadline implicated priority. The α adjusts the weight between them. The larger the α value is, the more important the application-specified priority becomes.
3) Unsent ratio. If the block is almost completed, then we would better finish its remaining part so that we do not waste the network resource.

The scheduler will run when ACK is received or the application pushes the data so that it can react fast to the network fluctuation and the change of the application's sending pattern. At each run, the scheduler updates the $Weighted_p$ values of the nondepending blocks and picks the block with

the lowest $Weighted_p$. If the calculation shows a block cannot meet the deadline ($remain_time < 0$), DTP tends to suspend sending but not drop it. A block will be dropped when it already missed the deadline on the sender side. In other words, dropped blocks are generally severely expired. When deadline setting is unreasonable which causes severe drops, the application needs to adjust the deadline.

The scheduler maintains a Directed Acyclic Graph (DAG) graph of blocks' dependencies. If there is a circular dependency, the graph will consolidate the circular depending blocks to a big block thus maint the acyclicity. Only blocks with zero in-degree (not dependent on any other blocks) are taken into consideration by the scheduling. A block is removed from the dependency graph once it is delivered or canceled.

C. Deadline-aware adaptive redundancy

The block picked by the ADS scheduler will be broken into a packet stream which will go through the redundancy module to get protection against loss. To reduce the computation and bandwidth overhead without losing redundancy protection, we design the redundancy module to be adaptive to the application sending pattern and the network.

The redundancy module is activated only for those packets which cannot afford retransmission delay. It is possible for a single block that part of the packets carrying its data trigger redundancy while others do not. The loss detection of a packet takes up one RTT (three duplicate ACKs) and the retransmission of the packet takes up one RTT. When a packet has less than two RTTs to meet its deadline, the redundancy module will be activated. Then the redundancy module will encode n redundant packets for following m consecutive packets using a block-based Forward Error Correction (FEC) scheme. These $m + n$ packets form a redundancy group. If there are not enough m packets from the application, we will use empty packets as original ones. Receiving any m packets of the group can decode all m original packets. We send redundant packets after original packets so that the encoding/decoding process can run in parallel with normal packet sending/receiving. The performance of redundancy module depends on the choice of m and n. Larger m and n can get better protection but also introduce higher overhead. The empty packets are also another overhead. We strategically set and adjust m, n to accomplish the restoration before the deadline without much overhead.

A larger m can provide better protection against burst loss but introduce higher overhead; also the decoding delay may be higher because receiver needs to wait at most m packets to decode. If the application has less than m packets to send, then the rest of original packets will be empty packets, wasting bandwidth. We choose m to be the biggest value satisfying that every packet in the group will not miss its deadline, *i.e.,* $\frac{m \times MSS}{bandwidth} + 2 \times RTT < deadline$. If more than half of the original packets are empty, we reduce the m by half. Another choice to address the problem is to avoid sending those empty packets but attach a flag with the last original packet. However, the loss of this packet will cause decoding to fail. It will make the protocol more complex to achieve the original protection effectiveness.

A larger n can also provide better protection against loss but will incur a higher overhead. Ideally, n should be able to cover the loss rate, *i.e.,* $n > m * loss_rate$. However, the loss rate measurement may be inaccurate and the loss pattern (bursty or random) is unknown. We choose n based on the redundancy group decoding result during the last 8 RTT (8 RTT is also the duration of a round of BBR cycle). When a redundancy group is finished (all packets inside it are either lost or ACKed), we check the packet loss rate of each group. Then we set n to cover the maximum loss rate over the last 8 RTT. Since the redundancy only applies to the close-to-deadline packets, there may not be any redundancy group finished during the cycle. In this case, we reset n using the loss rate measurement $n = m * loss_rate$. Picking the maximum loss rate will make n slightly bigger than the ideal value but will increase the chance of successful decoding.

IV. Implementation

We build DTP by extending QUIC because QUIC already provides many useful building blocks. Designs of DTP can also be implemented on other protocols such as SCTP and TCP. Our implementation is based on Cloudflare's quiche [24] which implements the IETF QUIC using Rust programming language. The implementation is split into the core QUIC state machine which has no I/O and socket management which can be written using platform-native API. In this way, the implementation can run on many platforms. Since QUIC is a protocol under fast development, our extension should be non-intrusive so that we can quickly follow the new features of QUIC and it is easier for those applications using QUIC to adopt DTP.

A QUIC packet is composed of a header and one or several frames which carry control information and application data. QUIC defines 20 types of frames such as STREAM, ACK, HANDSHAKE_DONE *etc.* The natural way to extend QUIC is to change its frame format or define a new one. To sync the block metadata between the receiver and sender, DTP defines two new frame types: BLOCK_INFO and BCT. To estimate one-way delay, DTP adds timestamp in ACK frame. FEC frame is added to implement the redundancy module. To make the deployment of DTP incremental, the version negotiation and transport parameter advertisement during handshake is modified to negotiate DTP features. If the negotiation fails, it will fall back to the vanilla QUIC.

A. Block-based delivery

DTP extends the QUIC to support the block-based delivery. Each block has its own deadline and priority. And some blocks may get dropped by the sender during transmission. QUIC only supports delivering multiple streams reliably. We need to find a way to map the block semantic to stream in QUIC. A QUIC implementation is required by QUIC RFC [13] to support the stream prioritization feature. Based on that, an intuitive way of mapping block to stream is to put blocks with

the same priority into one stream. In this way, we can reuse most parts of a QUIC scheduler for prioritization without any modification of QUIC wire format. However, this mapping faces an inherent problem: QUIC's reliable byte stream nature conflicts with the timeliness of dynamic block delivery. Suppose that one block in the middle of the stream passes its deadline and is decided to be dropped, then the sender needs to mark the specific bytes range as dropped, which means the stream delivery is no longer reliable. This will break the ack processing procedure and stream management, which makes the extension complicated. Existing extensions to add unreliable or partially reliable transmission to QUIC cannot accomplish that either. Tiesel *et al.* [25] propose a DATAGRAM frame to carry application data without requiring retransmission. Using DATAGRAM alone is not enough to implement block semantics since it does not support stream multiplexing.

We take another approach: mapping block to QUIC stream one to one. If the block is decided to be dropped, we just utilize the standard stream cancellation process of the existing QUIC protocol, which is sending the RESET_STREAM frame to cancel the stream. Using this mapping, we can reuse the max stream data size as the max block size. The block id can be mapped to the stream id without breaking the stream id semantic using Eq. (4). Since QUIC stream id is encoded using variable-length integer and the max value is $(2^{62} - 1)$. It is enough to hold block numbers during one session. Even if the block number is running out, we can easily process the wrap-around. By using the existing FIN bit of the STREAM frame [13], the receiver can easily determine whether the block transmission is finished. Unless mentioned, the DTP block features such as multiplexing are the same as the QUIC stream.

$$stream_id = (block_id << 2) \mid stream_type_bits \quad (4)$$

B. Metadata sync

As discussed in Section III-A, DTP needs to sync some information between the sender and receiver. First, DTP extends the ACK frame in QUIC with the packet received timestamp of the last packet within each packet range. Second, the block size is unknown to the receiver until the last chunk of the block arrives since QUIC is a stream-oriented protocol. DTP adds a new frame type BLOCK_INFO to convey the block metadata such as block size, block beginning time, deadline and priority to receiver. Each field is optional and encoded using a variable-length integer (1-4 bytes). It is sent prior to any block data frame. Third, DTP adds a new frame type BCT to convey the time difference between the expected and actual block completion time. When the receiver side application tells the difference to DTP, it will send a BCT frame back to sender. Both BLOCK_INFO and BCT frames are transmitted reliably, which means they will trigger ACK.

C. Redundancy

DTP follows a similar approach as QUIC-FEC [26] to use the clear text packet payload as a source symbol. Instead of supporting multiple FEC schemes, we choose the Reed-Solomon codec scheme to implement the redundancy module because it strikes the balance between computational overhead and efficiency. By fixing the scheme, we can make a few simplifications based on [26].

The first simplification is that DTP can avoid splitting the repair symbol by carefully picking the source symbol size. QUIC-FEC [26] designs the FEC frame to contain an offset field to handle the case when the repair symbol is bigger than a single FEC frame. Doing so may result in the repair symbol being split into two QUIC packets, making the decoding time hard to estimate. Since the repair symbol's size is the same as the source symbol size, if we consider the possible overhead when picking the clear text payload of a packet as the source symbol, we can avoid the repair symbol split across packets.

The source symbol size picking process is as follows. Normally, QUIC packet size is a constant value (MSS). There are a few edge cases that payloads may have different sizes. The first one is that packet headers have different lengths. QUIC uses a short header format for application packets after the handshake. The only field with varying length is the packet number field encoded using variable-length integers (can be 1/2/4 bytes). When the packet number is about to cross the length border, we leave the extra space when filling the payload with block data so that the size of the payload across the border is unified. The second one is when the last chunk of a block cannot fill the packet. If the next block uses the same redundancy rate, we bundle the data together. If not, we fill the payload with the PADDING frame which is naturally understood by QUIC protocol. We modify the FEC frame to contain the metadata of the redundancy group such as group id, index of the payload inside the group, m, n and the symbol.

The second simplification is that DTP stops the retransmission without defining any new frame type. QUIC-FEC [26] proposes a new RECOVERED frame containing the packet number which has been successfully recovered. For the receiver to know the recovered packet number, the packet number of each packet needs to be included in the source symbol which will take extra space. The header protection process [27] will mask the packet number field, which makes the encoding more complicated. DTP avoids the RECOVERED frame by leveraging the property of Reed-Solomon codec to derive the recovered event from ACK at the sender without doing the actual decoding. Consider a redundant group that contains m original packets and n redundant packets. Since QUIC packets are authenticated, receiving any m packets means the whole group can be recovered. DTP requires the block-based codec to support this simplification and does not support sliding window solutions such as RLC. The successful running of DTP depends on enough buffer for the codec scheme.

D. Handling clock synchronization error

Using the timestamp of the sender and receiver to get the block completion time and one-way delay requires clock synchronization between hosts. NTP [28] can usually maintain time to within a few milliseconds [29] over the Internet

```
Sender:
int send(int fd, void *block, size_t size,
    int id, int deadline, int priority,
    int *depending_ids, size_t depending);
int update(int fd, int id,
    int priority, int deadline,
    int *depending_ids, size_t depending);
int cancel(int fd, int id);
void on_dropped(int fd,int id, int priority,
        int deadline, int goodbytes);
void on_delivered(int fd, int id, int priority,
        int deadline, int delta, int goodbytes);
Receiver:
int recv(int fd, void *block, size_t size, int id);
int expect(int fd, int id, int delta);
```

Fig. 2: Interfaces of DTP

which is good enough for DTP. But the error can reach 100ms or more in extreme cases. DTP addresses this problem by leveraging knowledge from application. When a block misses its deadline, the QoE of the application will be affected and receiver side application will notice that (video stream stall, game sync missing this round). If the missing happens consistently, then the deadline setting is unrealistic under current network (either because of long RTT or clock synchronization error). The sender is supposed to adjust its deadline requirement.

Another possible situation is the endpoint(s) do not sync time. DTP uses $system_error$ parameter to find and resolve this problem. We assume that when the RTT is stable, the one-way delay is stable too. For each packet, the packet arrival time is $send_time + one_way_delay + system_error$. $system_error$ includes the error of the one-way delay estimation and the time sync inaccuracy between the sender and receiver. The sender compares the real and estimated arrival time when the RTT is stable to adjust the $system_error$. When RTT changes, the $system_error$ will be re-calibrated. This method is similar to NTP's basic interaction mechanism and achieves close precision in our practice.

E. Abstraction and API

We extend the socket API to allow the application attaching metadata along with the data block, as is shown in Fig. 2. The fd is the connection id. The $depending_ids$, $depending$ is the array of ids of block which current block depends on directly. The $update$ and $cancel$ function is used to update the metadata of the block or cancel the block. We design $on_dropped$ and $on_delivered$ to report the delivery result of each block. The amount of data delivered before deadline is stored in $goodbytes$. The difference between block completion time and the deadline in milliseconds is stored in $delta$. Receiver side application can use $expect$ to tell DTP the difference between expected and actual block completion time.

DTP provides in-order and reliable delivery within the block but does not guarantee all blocks will be delivered before the deadline due to network limitations. For each block, DTP provides the delivery result such as block completion time when the block is completed and block completion ratio when a block is dropped. The application can use delivery results to optimize its performance.

V. Developing application upon DTP

In this section, we will discuss how applications can use DTP. To use DTP, the application needs to determine the deadline, priority, and dependency of each block. The choice of these parameters should base on the application-specific mechanisms that affect QoE. For example, a conference application could prioritize audio frames higher and set deadlines based on acceptable communication delays or jitter buffer size. We develop two applications using DTP shown in Table II.

A. 360° video streaming emulation

We build a 360° video streaming emulation. The emulation framework is similar to Figure 2 of Flare [17] without the actual decoding and display. The app uses the user's head movement trace to predict the future head movement and converts that to viewport and requests tiles inside the viewport from the server. Each block has several different quality versions. To choose the quality version, the app runs a simple ABR algorithm that aggregates the network capacity using $on_delivered$ callback over the last second and chooses the highest quality version allowed by the deadline of blocks.

B. Video conferencing

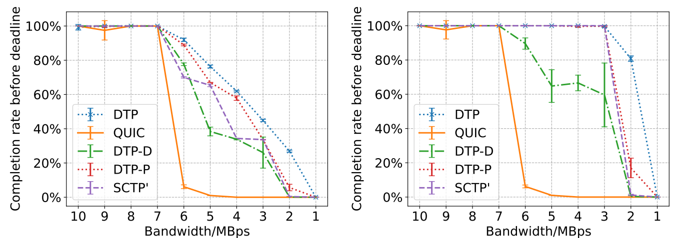
We implement a lightweight video conferencing application using DTP and FFMPEG. The server sends multiple video streams to the app. The server divides the video stream into a series of frames. Each frame inside the same video stream has the same priority. The app will put these blocks into each stream's jitter buffer. Video decoder will wait for the jitter buffer to contain at least 100 ms data to begin its decoding. The application determines what to transfer and DTP determines how to transfer it. The codec and transport protocol is not as coupled as in WebRTC. We implement it based on TCP, considering TCP is still the most widely used protocol. Thanks to the similarity between DTP's API and BSD sockets and compatibility with popular event-driven frameworks, we only modify 98 lines of code out of 5K LoC to replace TCP with DTP. Interfaces of many other existing protocols such as UDP and SCTP are also close to or compatible with the socket API, thus the migration cost will not be too great.

VI. Evaluation

In this section, we evaluate DTP under various network conditions and different applications. Unless otherwise specified, experiments in this section are conducted between two PCs (Intel Core i5 3.4GHz, 8GB memory) running Ubuntu 18.04 communicate through emulated network. The clock time of two PCs is synchronized. We run tc [30] in a separate router to simulate various network conditions since we find similar erroneous results as Kakhki et al. [31] when using

TABLE II: Deadline-aware applications using DTP

Application	**Block**	**Deadline**	**Priority**	**Dependency**
360° video streaming	Tile	Predicted playback time	Area inside viewport	No
Video conferencing	Frame	100 ms	Activeness of the stream	Frame dependency



(a) Blocks' deadline meeting rate (b) High priority blocks' deadline meeting rate

Fig. 3: Various static bandwidth

tc at endpoint. We compare DTP with QUIC and SCTP. For the QUIC competitor, we select the default setting of the quiche release DTP based on. We use no reset operation to drop streams of QUIC, because the application cannot know when to cancel a stream without information maintained and provided by the DTP. We implemented the default SCTP scheduler (First-come, First-serve) with cancel of stale data based on QUIC (Denoted SCTP'). DTP, QUIC and SCTP' are running the same BBR congestion control algorithm and block-to-stream mapping for apple-to-apple comparison. For DTP, unless otherwise specified, we set $\alpha = 0.5$ to equally consider deadline and priority, and set β as the lowest priority value, which means a tendency to give up about-to-miss-deadline blocks when there are fresh ones.

We first study the behavior of DTP using artificial simple network and application traces (Section VI-A), and then demonstrate the performance under more realistic scenarios (Sections VI-B and VI-C).

A. *Transport layer performance*

First, we evaluate the transport layer performance using a dummy app so that we can test different traffic patterns under different network conditions. We measure the CPU time used by the dummy app, and find no observable difference in CPU consumption between the DTP-based and QUIC-based version when sending the same amount of data. In the following experiments, we consider 3 variants of DTP: (1) only consider deadline ($\alpha = 0$, denoted DTP-D); (2) only consider priority ($\alpha = 1$, denoted DTP-P); (3) consider both deadline and priority (denoted DTP). We run all tests 10 times. Each test is running for one minute.

1) Bandwidth variance: We first investigate how DTP performs under varying bandwidth conditions. The app sends three blocks of 200 KB with distinct priorities every 100 ms. The app sorts the sending order of blocks based on their priorities. The deadline for each block is 200 ms. The bandwidth requirement for all blocks to arrive before the deadline is 6 MBps. We vary the bandwidth between 1 MBps and 10 MBps. We set the RTT to 5 ms and the random loss rate to 0.01%. Fig. 3a shows the ratio of blocks that are delivered before the deadline. Note the bandwidths marked on the horizontal axis are the emulated values using tc. The achieved throughput is always slightly smaller than the set value, even when we use iperf [32] to send enough UDP packets. When the bandwidth is sufficient ($> 6M$), all blocks can arrive before the deadline. When the bandwidth is limited, QUIC's performance degrades fast because it does not drop stale blocks.

DTP-D and DTP-P perform slightly worse than DTP but way better than QUIC. This proves that considering the deadline or priority can improve the deadline delivery performance and combining those two factors can get even better performance. DTP-D performs worse than DTP-P because only considering the deadline will result in lots of preemption. Take the 5 MBps case as an example, the DTP-P scheduler will send the first two high priority blocks and they can complete before the deadline. However, the DTP-D scheduler will still choose to send the block with the lowest priority when it becomes the most close-to-deadline block. After sending the block for a while, the scheduler will find other blocks closer to deadline and it will switch again, wasting the bandwidth to transfer the lowest-priority block.

When bandwidth is 5/3 MBps (which is only enough to finish two/one of three blocks before deadline each round), the performance of SCTP' is the same as DTP-P. Recall that the app sorts the sending order of blocks based on their priority each round. Both SCTP' and DTP-P will finish the first two/one blocks each round and cancel the remaining blocks. When bandwidth is 6/4 MBps, the achieved throughput is smaller than 6/4 MBps. SCTP' performs worse than DTP-P. When a new round of block arrives, SCTP' will continue to send old blocks until they miss their deadline but DTP-P will switch to higher priority blocks immediately. Only when higher priority/new blocks finish and old blocks are still not canceled, DTP-P will consider $unsent_ratio$ and finish the block which has less remaining data. This demonstrates that strategic preemption based on priority and $unsent_ratio$ can improve the performance.

When bandwidth is 6 MBps, the first two blocks of each round can finish before deadline. As for the third block, DTP-D will predict that it will miss the deadline, so it will not send it. But SCTP' will continue to send the old one until it misses deadline. This demonstrates that proactively giving up the about-to-miss-deadline blocks can reduce the bandwidth waste thus get better performance. Combining all above factors, DTP consistently performs best.

We measure the data that has arrived before the deadline (good bytes). Then we divided the good bytes by total bytes to get the good bytes ratio. Most bytes sent by QUIC miss their deadline while bytes from all other four competitors can reach about 100% good bytes ratio. This indicates that DTP can make better use of bandwidth to deliver fresh data.

We also measure the complete-before-deadline ratio of

blocks with the highest priority. The result is shown in Fig. 3b. SCTP' behaves like DTP-P except when the bandwidth is 2 MBps. In this case, the bandwidth requirement of the highest block is 2 MBps and actual throughput is smaller than 2 MBps. DTP will choose to give up old blocks but DTP-P and SCTP' will stick to them. Therefore, DTP performs better than DTP-P and SCTP'. DTP-D will switch between different blocks when the bandwidth is smaller than the total required (6 MBps). Therefore, it performs worse than DTP-P, DTP and SCTP'.

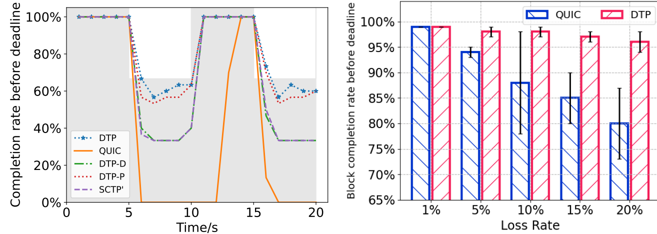


Fig. 4: Deadline meeting rate under dynamic bandwidth

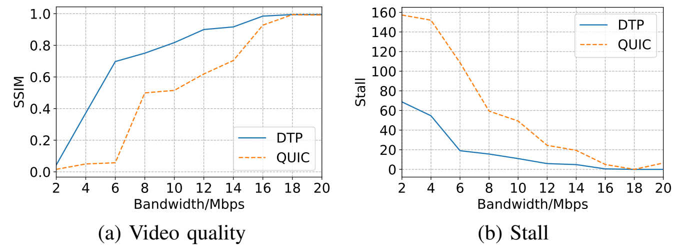
Fig. 5: Deadline meeting rate under different loss rates

To further understand where the improvement comes from, we measure the average waiting time at the sender of each completed block (figure omitted due to space limitation). QUIC does not drop and the sender queue will accumulate when lacking enough bandwidth. The DTP reduces the waiting time without the sacrifice of blocks' deadline meeting. None of the variants can achieve such balance, either sacrificing the low priority blocks completion rate, or repeatedly switching and making the delay at sender larger.

To evaluate how DTP reacts to the dynamic bandwidth, we vary the bandwidth during one test. The bandwidth is reduced suddenly at 5 seconds from 12 MBps to 4 MBps and restored at 10 seconds (indicated by the grey area in Fig. 4). Each point represents the block completion rate in the past second. Similar to the static bandwidth results, DTP consistently performs best.

2) Loss variance: We then investigate the performance of DTP when facing network loss. We evaluate the effectiveness of the redundancy module under three different traffic patterns. For each scenario, we vary the loss rate from 1% to 20%. We set the bandwidth to be big enough for all sending blocks.

The first traffic pattern is sending three blocks of 100 KB with distinct priority each 100 ms. The RTT is 20 ms. In this scenario, thanks to adaptive redundancy, only the tail data needs to be protected. We verify that by counting the additional data sent. For example, only about 3% extra data is sent by the redundancy module when the loss rate is 10%. For the second and third traffic patterns we simulate online gaming. Each gaming command is very small, usually smaller than MSS. Therefore, we set the block size to 1 KB. We set the RTT to be 60 ms so that any packet lost will result in the block missing its deadline. All packets will get redundancy protection. The results are similar. We only show the sparse small blocks' result in Fig. 5 due to space limitation. DTP constantly outperforms QUIC under all scenarios. This shows that the redundancy module is capable of handling diverse application traffic patterns and losses.



(a) Video quality

(b) Stall

Fig. 6: Video conferencing performance

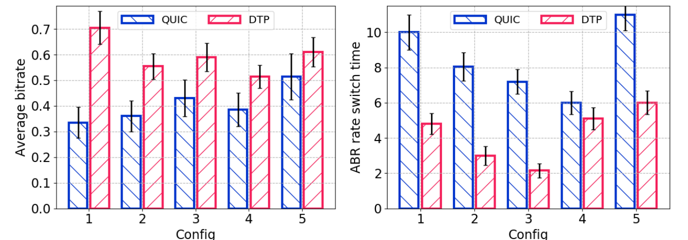
B. QoE of applications

In this section, we evaluate two applications built on top of DTP. The baseline is using QUIC instead of DTP.

1) Video conferencing: We feed 8 different video streams into the video conferencing server. The bandwidth is varied from less than the smallest stream to bigger than 8 streams combined. The loss rate is set to 0.01% and the RTT is set to fluctuate between 50 to 90 ms. Those 8 streams are shown as 8 squares on the receiver screen. The active stream (with higher priority) will have a bigger area on the receiver's screen thus a bigger impact on image quality. We count the stall of the highest stream as a stall for the application. As is shown in Figs. 6a and 6b, DTP can reduce stall and increase the video quality because it can increase the highest priority block completion rate. When the bandwidth is set to 6 Mbps, DTP can achieve over 5x video quality than QUIC.

2) 360° video: We run our 360° video emulator using a wide variety of real traces. We use the user head movement trace from [17]. We use the head movement over the last three seconds to do a linear prediction for the next one second. Our 360° video traces contain car tracing, concert, wild animal and skydive. The network traces contain 3G, 4G, Hotel Wi-Fi and are scaled to match the video bitrate. We randomly pick five combinations of those traces as the configuration for the test. Each test runs for five minutes and is repeated 10 times.

We first measure the average bitrate of tiles inside the viewport (Fig. 7a). Since DTP drops low priority block (out of sight tiles), more bandwidth is allocated to tiles inside viewport thus DTP can achieve higher bitrate than QUIC. We also measure the stall time per unit time. DTP achieves lower stall time than QUIC (figure omitted due to space limitation). Finally we measure the bitrate change frequency (Fig. 7b). Since DTP has no send buffer accumulation, it can give more smoothed network feedback to the application. As a result, ABR rate change happens less frequently in DTP than QUIC.



(a) Average bitrate of tiles inside viewport (Higher is better)

(b) ABR rate change frequency (Lower is better)

Fig. 7: QoE of 360-degree video streaming

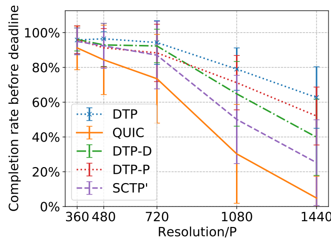


Fig. 8: Performance of varying video traces under cellular

C. Real-world network evaluation

To test the performance under real-world network, we conducted two tests over WLAN or cellular network.

1) Application traffic pattern over cellular network: We conduct the video streaming experiment between a desktop client, tethered to commercial 4G network, and an Ali Cloud server located in Asia. We use various video streaming traces to evaluate how DTP reacts to different block sizes and dynamic network. Fig. 8 shows results for these tests. The results are similar to transport layer test under varying bandwidth (Fig. 3a), except the DTP-D performs better than SCTP'. This potentially from that timely preemption provides more benefit under changing network.

TABLE III: Real world QoE improvement over QUIC

Scenario	Home		Restaurant	
QoE metric	Bitrate	Stall	Bitrate	Stall
Improvement	13%	30%	15%	43%

2) 360° video over WLAN: We run the 360° simulator between a laptop and AWS server located in North America. We run the test from home and restaurants. When running in the home, other computer in the same WLAN is running a bandwidth-hungry application such as streaming 4K video. In this way, we can test the performance of DTP under other flows' interference. When the bottleneck resides in the network, the queue will accumulate at the sender. DTP can reduce the queue size at the sender, thus can improve the QoE as shown in Table III.

VII. Related work

Layer 2 and 3 techniques to improve the latency. Many efforts provide deterministic latency on layer 2 (TSN [4]) and layer 3 (DetNet [3]). They require modification to the switch and router hardware.

Deadline-aware solution requiring in-network cooperation. Many solutions provide deadline-aware delivery by cooperation between end hosts and in-network elements. For example, D3 [5] modifies the switch to allocate the rate based on the deadline. D2TCP [6] uses ECN in existing switches to adjust the sending rate to meet the deadline. The in-network cooperation makes these solutions more powerful in deadline meeting but also make them difficult to deploy on the Internet.

Traditional real-time application optimization. Salsify [7] co-design the codec and transport to proactively set the encoding rate to match the available bandwidth. It requires replacing the whole stack and does not support existing hardware encoder and decoder. AWStream [8] proposes a new API for the application to supply the rate adaptation function such as skip frames and reduce resolution. It uses offline profiling to learn the accuracy of adaptation methods. At runtime, it matches the sending rate and available bandwidth by using the learned configuration. These tightly cross-layer coupled solutions are complex and hard to port to emerging applications.

Extension to traditional end-to-end transport protocols. SCTP supports $life\ time$ parameter [10] for the message. Mukherjee *et al.* [11] modify TCP to only send data before the deadline but make it not wire-compatible with TCP. McQuistin *et al.* [12] extend the idea by using COBS encoding to make it wire-compatible with TCP. But the encoding introduces high overhead and complex modifications to the kernel.

These proposals are good steps toward a deadline-aware transport protocol. However, first, they do not do anything to improve the deadline delivery. Only dropping the stale data is not enough as show in Section VI. Second, they all suffer from deployment issues. SCTP and TCP are restricted by middleboxes [33] and kernel changes. Our proposal improves delivery before the deadline and can be deployed incrementally.

Preliminary version. In a published short paper [34], we studied deadline requirements of emerging applications and provided high-level solution directions about block-based delivery, deadline-aware scheduler and API of deadline-aware transport. In this work, we present critical advancements including the Active-drop-at-sender scheduler, adaptive redundancy, and protocol extensions on QUIC, including frame extending, block mapping and packetizer design. We also implement DTP based on QUIC and address problems like metadata sync. In addition, we employ DTP in two applications and extensively evaluate its performance and behavior.

VIII. Conclusion

In this paper, we propose a new transport-layer protocol called DTP. It employs two key mechanisms, deadline-aware Active-Drop-at-Sender (ADS) scheduling and adaptive redundancy, to boost the application performance. We build and evaluate several applications on top of DTP. Compared to vanilla QUIC, DTP can improve applications' QoE by 1.2x to 5x, while incurring very small burden on application developers.

Acknowledgment

This work is not possible without the efforts and support of Yuming Hu, Kewei Zhu, Zhiwen Liu, and Sijiang Huang. We thank Chuan Ma and Kunpeng He for their help and feedback. We thank our shepherd, Spyridon Mastorakis, and all anonymous reviewers for reviewing the previous versions of the paper and providing valuable comments.

This work was supported in part by the National Key R&D Program of China (No.2018YFB1800303) and National Nature Science Foundation of China (No.62132009).

REFERENCES

[1] M. Baldi and Y. Ofek, "End-to-end delay analysis of videoconferencing over packet-switched networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 8, no. 4, pp. 479–492, 2000.

[2] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, and N. Dai, "Furion: Engineering high-quality immersive virtual reality on today's mobile devices," in *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*. ACM, 2017, pp. 409–421.

[3] "Deterministic Networking Working Group, IETF," https://datatracker.ietf.org/wg/detnet/, 2020.

[4] "Time sensitive network," http://www.ieee802.org/1/pages/tsn.html, 2018.

[5] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 50–61, 2011.

[6] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter tcp (d2tcp)," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 115–126, 2012.

[7] S. Fouladi, J. Emmons, E. Orbay, C. Wu, R. S. Wahby, and K. Winstein, "Salsify: Low-latency network video through tighter integration between a video codec and a transport protocol," in *NSDI 18*, 2018.

[8] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, and E. A. Lee, "Awstream: Adaptive wide-area streaming analytics," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 2018, pp. 236–252.

[9] E. Kohler, M. Handley, and S. Floyd, "Datagram congestion control protocol (dccp)," Internet Requests for Comments, RFC Editor, RFC 4340, March 2006, http://www.rfc-editor.org/rfc/rfc4340.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4340.txt

[10] R. Stewart, "Stream control transmission protocol," Internet Requests for Comments, RFC Editor, RFC 4960, September 2007, http://www.rfc-editor.org/rfc/rfc4960.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4960.txt

[11] B. Mukherjee and T. Brecht, "Time-lined tcp for the tcp-friendly delivery of streaming media," in *Proceedings 2000 International Conference on Network Protocols*. IEEE, 2000, pp. 165–176.

[12] S. McQuistin, C. Perkins, and M. Fayed, "Tcp hollywood: An unordered, time-lined, tcp for networked multimedia applications," in *2016 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE, 2016, pp. 422–430.

[13] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," RFC 9000, May 2021. [Online]. Available: https://rfc-editor.org/rfc/rfc9000.txt

[14] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar *et al.*, "The quic transport protocol: Design and internet-scale deployment," in *Proceedings of the conference of the ACM special interest group on data communication*, 2017, pp. 183–196.

[15] X. Liu, Q. Xiao, V. Gopalakrishnan, B. Han, F. Qian, and M. Varvello, "360 innovations for panoramic video streaming," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. ACM, 2017, pp. 50–56.

[16] F. Qian, L. Ji, B. Han, and V. Gopalakrishnan, "Optimizing 360 video delivery over cellular networks," in *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*. ACM, 2016, pp. 1–6.

[17] F. Qian, B. Han, Q. Xiao, and V. Gopalakrishnan, "Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. ACM, 2018, pp. 99–114.

[18] J. He, M. A. Qureshi, L. Qiu, J. Li, F. Li, and L. Han, "Rubiks: Practical 360-degree streaming for smartphones," in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2018, pp. 482–494.

[19] T. Kämäräinen, M. Siekkinen, J. Eerikäinen, and A. Ylä-Jääski, "Cloudvr: Cloud accelerated interactive mobile virtual reality," in *2018 ACM Multimedia Conference on Multimedia Conference*. ACM, 2018, pp. 1181–1189.

[20] L. Liu, R. Zhong, W. Zhang, Y. Liu, J. Zhang, L. Zhang, and M. Gruteser, "Cutting the cord: Designing a high-quality untethered vr system with low latency remote rendering," in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2018, pp. 68–80.

[21] H. Qiu, F. Ahmad, R. Govindan, M. Gruteser, F. Bai, and G. Kar, "Augmented vehicular reality: Enabling extended vision for future vehicles," in *Proceedings of the 18th International Workshop on Mobile Computing Systems and Applications*. ACM, 2017, pp. 67–72.

[22] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control," *Queue*, vol. 14, no. 5, pp. 20–53, 2016.

[23] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "Tcp vegas: New techniques for congestion detection and avoidance," in *Proceedings of the conference on Communications architectures, protocols and applications*, 1994, pp. 24–35.

[24] "quiche: Cloudflare's implemention of ietf quic," https://github.com/cloudflare/quiche, 2021.

[25] P. Tiesel, M. Palmer, B. Chandrasekaran, A. Feldmann, and J. Ott, "Considerations for unreliable streams in quic," Working Draft, IETF Secretariat, Internet-Draft draft-tiesel-quic-unreliable-streams-01, October 2017. [Online]. Available: http://www.ietf.org/internet-drafts/draft-tiesel-quic-unreliable-streams-01.txt

[26] F. Michel, Q. De Coninck, and O. Bonaventure, "Quic-fec: Bringing the benefits of forward erasure correction to quic," in *2019 IFIP Networking Conference (IFIP Networking)*. IEEE, 2019, pp. 1–9.

[27] M. Thomson and S. Turner, "Using tls to secure quic," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-quic-tls-25, January 2020, http://www.ietf.org/internet-drafts/draft-ietf-quic-tls-25.txt. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-quic-tls-25.txt

[28] D. L. Mills *et al.*, "Network time protocol (ntp)," 1985.

[29] "chrony: a versatile implementation of NTP," https://chrony.tuxfamily.org/.

[30] "tc: Linux Advanced Routing and Traffic Control," http://lartc.org/lartc.html.

[31] A. M. Kakhki, S. Jero, D. Choffnes, C. Nita-Rotaru, and A. Mislove, "Taking a long look at quic: an approach for rigorous evaluation of rapidly evolving transport protocols," in *Proceedings of the 2017 Internet Measurement Conference*, 2017, pp. 290–303.

[32] A. Tirumala, "Iperf: The tcp/udp bandwidth measurement tool," *http://dast. nlanr. net/Projects/Iperf/*, 1999.

[33] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend tcp?" in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, 2011, pp. 181–194.

[34] H. Shi, Y. Cui, F. Qian, and Y. Hu, "Dtp: Deadline-aware transport protocol," in *Proceedings of the 3rd Asia-Pacific Workshop on Networking 2019*, ser. APNet '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–7. [Online]. Available: https://doi.org/10.1145/3343180.3343191