Workgroup: QUIC
Internet-Draft: draft-shi-quic-dtp-09
Published: 28 January 2024
Intended Status: Informational
Expires: 31 July 2024
Authors: Y. Cui
Tsinghua University
C. Ma
Tsinghua University
H. Shi
Huawei
K. Zheng
Huawei
W. Wang
Huawei

Deadline-aware Transport Protocol

Abstract

This document defines Deadline-aware Transport Protocol (DTP) to provide block-based deliver-before-deadline transmission. The intention of this memo is to describe a mechanism to fulfill unreliable transmission based on QUIC as well as how to enhance timeliness of data delivery.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-shi-quic-dtp/.

Source for this draft and an issue tracker can be found at https://github.com/STAR-Tsinghua/DTP-draft.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 31 July 2024.

Copyright Notice

Table of Contents

1. Introduction

Many emerging applications have the deadline requirement for their data transmission. However, current transport layer protocols like TCP [RFC0793] and UDP [RFC0768] only provide primitive connection establishment and data-sending service. This document proposes a new transport protocol atop QUIC [QUIC] to deliver application data before end-to-end deadline.

1.1. Conventions and Definitions

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Motivation

Many applications such as real-time media and online multiplayer gaming have requirements for their data to arrive before a certain time i.e., deadline. For example, the end-to-end delay of video conferencing system should be below human perception (about 100ms) to enable smooth interaction among participants. For Online multiplayer gaming, the server aggregates each player's actions every 60ms and distributes these information to other players so that each player's state can be kept in sync. Data missing the deadline is useless since it will be overwrote by the new data.

These real-time applications have following common features:

*They tend to generate and process the data in block fashion. Each block is a minimal data processing unit. Missing a single byte of data will make the block useless. For example, video/audio encoder produces the encoded streams as a series of block(I,B,P frame or GOP). Decoder consumes the frame into the full image. For online games, the player's commands and world state will be bundled together as a message.

*They will continuously generate new data. Different from web browsing or file syncing, real-time applications like video conferencing and online multiplayer gaming have uninterruptedly interactions with users, and each interaction requires a bunch of new data to be transmitted.

*They prefer the timeliness of data instead of reliability since data missing deadline are useless to application and will be obsoleted by newer data. For example in multiplayer online games, the gaming server will broadcast the latest player states to every client, and the old information does not matter if it can

not be delivered in time. So the meaningful deadline of the application is actually the block completion time i.e., the time between when the block is generated at sender and when the block is submitted to application at receiver.

However, current transport layer protocols lack support for block-based deadline delivery. TCP guarantees reliability so it will waste network resource to transmit stale data and cause fresh data to miss its deadline. UDP is unreliable but it doesn't drop data according to deadline, all data have the same chance to be dropped indeed. QUIC makes several improvements and introduces Stream Prioritization [QUIC] to enhance application performance, but prioritization is not enough for enhancing timeliness.

Insufficiency of existing transport layer forces applications to design their own customized and complex mechanism to meet the deadline requirement. For example, the video bitrate auto-adjustment in most streaming applications. But this is a disruption to the Layered Internet Architecture, forcing applications to worry about network conditions.

This document proposes Deadline-aware Transport Protocol (DTP) to provide deliver-before-deadline transmission. DTP is implemented as an extension of QUIC (Refer to Section 4) because QUIC provides many useful features including full encryption, user space deployment, zero-RTT handshake and multiplexing without head-of-line blocking.

3. Design of DTP

The key insight of DTP is that these real-time applications usually have multiple blocks (As shown in Figure 1 below) to be transferred simultaneously and these blocks have diverse impact on user experience(denoted as priority). For example, audio data is more important than video stream in video conferencing. Central region is more important than surrounding region in 360 degree video. Foreground object rendering is more important than the background scene in mobile VR offloading.

The priority difference among multiple blocks makes it possible to drop low priority data to improve timeliness of high priority data delivery, which can enhance the overall QoE if resources allocated to blocks are correctly prioritized. In this section, we describe the mechanism which enables DTP to leverage that insight.

3.1. Abstraction

DTP provides block-based data abstraction for application. A 'block' is a piece of continuous data. A partial delivered block is useless for applications, and each block can be independently processed.

Application **MUST** attach metadata along with the data block to facilitate the scheduling decision, those metadata include:

- Each block has a deadline requirement, meaning if the block cannot arrive before the deadline, then the whole block may become useless because it will be overwrote by newer blocks. The application can mark the deadline timestamp indicating the deadline of its completion time. In the API of DTP, the deadline argument represents the desired block completion time in ms.

- Each block has its own importance to the user experience. The application can assign each block a priority to indicate the importance of the block. The lower the priority value, the more important the block. The priority argument also indicates the reliability requirement of the block. The higher priority, the less likely the block will be dropped by sender.

The sender can actively drop any block. DTP **SHOULD** transmit every undropped block reliably.

3.2. Architecture of DTP

The sender side architecture is shown in Figure 1:

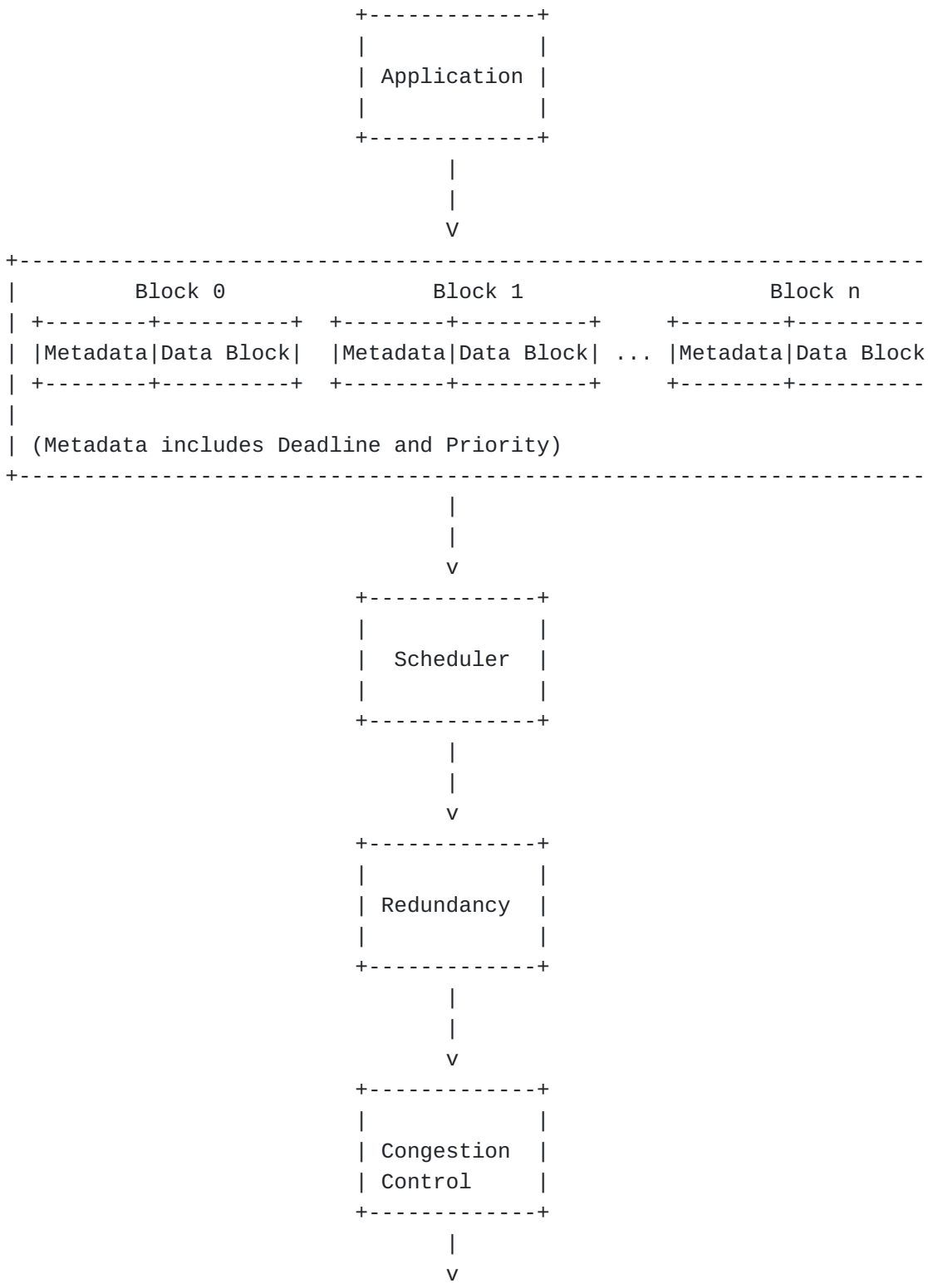

Figure 1: The Architecture of DTP

3.3. Deadline-aware Scheduler

The scheduler will pick the blocks to send and drop stale blocks when the buffer is limited. This section describes the algorithm of DTP scheduler.

Scheduler of DTP takes into account many factors when picking blocks in sender buffer to send. The goal of the scheduler is to deliver as much as high priority data before the deadline and drop obsolete or low-priority blocks. To achieve this, the scheduler utilizes both bandwidth and RTT measurement provided by the congestion control module and the metadata of blocks provided by the application to estimate the block completion time. The scheduler will run each time ACK is received or the application pushes the data.

A simple algorithm which only considers priority cannot get optimal result in transmitting deadline-required data. 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. But it should do so without causing the high priority stream missing the deadline. This example reveals a fundamental conflict between the application specified priority and deadline implicated priority. DTP needs to take both priorities into consideration when scheduling blocks.

DTP will combine all these factors to calculate real priority of each block. Then the scheduler just picks the block with the highest real priority. Scheduler of DTP will calculate the block remaining transmission time and then compare it to the deadline. The closer to the deadline, the higher real priority. And higher application specified priority will also result in higher real priority. In this way, the scheduler can take both approaching deadline and application-specified priority into account. Blocks which are severely overdue can be dropped accordingly.

3.3.1. Block dropping mechanism

DTP allows the sender side to cancel sending several blocks in the transport layer, and this action is called 'drop'. By dropping some stale blocks, DTP can enhance the timeliness of other sending blocks and save bandwidth. DTP **SHOULD** implement some strategies on the sender side to determine which 'block' should be dropped. On the receiver side, DTP **SHOULD** be able to check which block is dropped and **MAY** have functions to inform the application about the canceled blocks.

3.4. Deadline-aware Redundancy

After the scheduler pick the block to send, the packetizer will break the block into packet streams. Those packet streams will go through the redundancy module. When the link is lossy and deadline is tight, retransmission will cause the block missing the deadline. Redundancy module has the ability of sending redundancy (like FEC Repair Symbols) along with the data that will help to recover the data packets (like FEC Source Symbols), this can avoid retransmission.

We use unencrypted DTP packets as input to Redundancy Module because the loss of a DTP packet exactly corresponds to the loss of one Redundancy Packet. And to perform the coding and decoding with packets of different sizes, some packets may need to be padded with PADDING Frame. The present design of Redundancy Module follows the FEC Framework specified in [arXiv_1809.04822]. Figure 2 illustrates this framework:

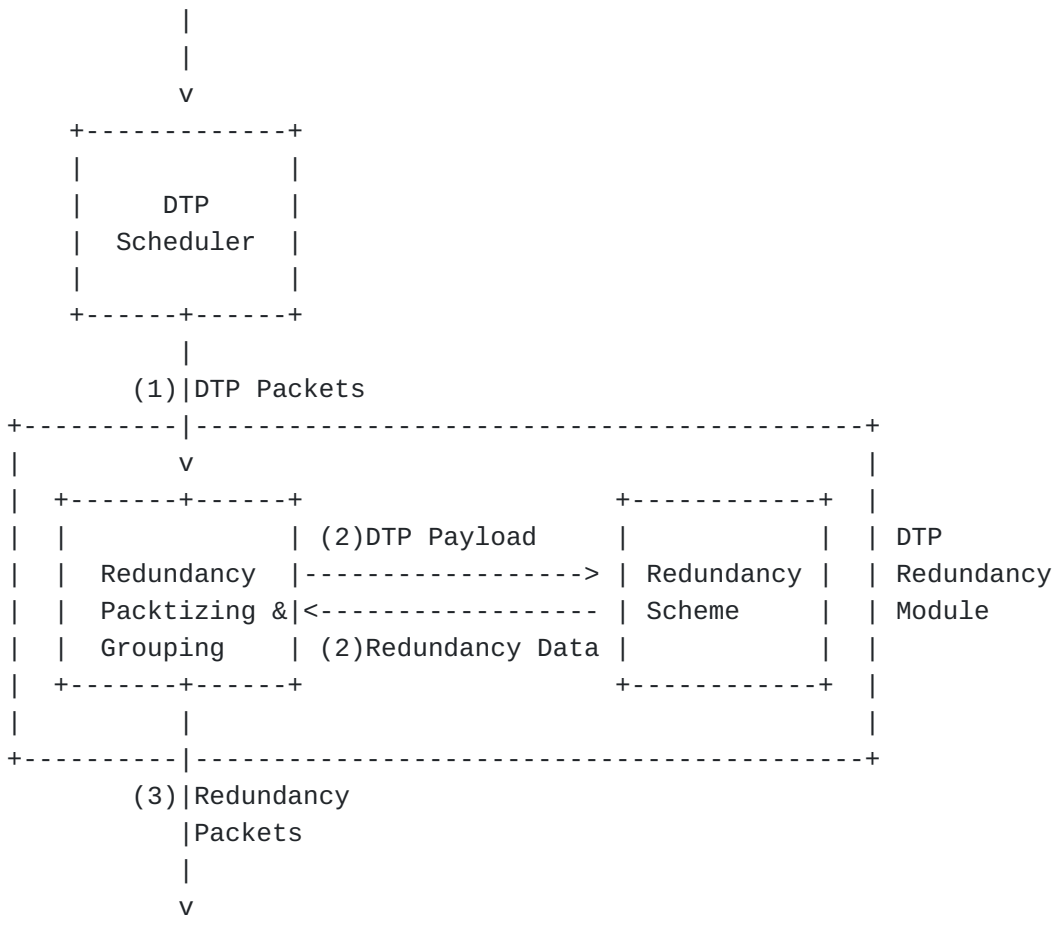

Figure 2: DTP Redundancy Module

Figure 2 above shows the mechanism of how the Deadline-aware Redundancy module works. (1) Redundancy Module first receives the unencrypted DTP packets from scheduler. (2) The Redundancy Scheme use DTP Payload (similar to FEC Repair Symbols) to generate Redundancy Data (similar to FEC Source Symbols). (3) Redundancy-protected DTP Packets and Redundancy Data will be packtized and grouped. Redundancy Packtizing and Grouping Part will generate FEC Payload INFO (Figure 6) and attach it to the DTP Packets and Redundancy Data, generating Redundancy Packets (a Redundancy Packet with the header shown in Figure 6). Once the protocol receives the Repair Symbols, they are sent to the receiver through the FEC Packets. At the receiver-side, the received Redundancy Packets can be processed immediately. The Redundancy Data is reconstituted from the Redundancy Packtizing and Grouping and passed to the underlying Redundancy Scheme to recover the lost DTP Packets.

Although Redundancy Module allows recovering lost packets without waiting for retransmissions, it consumes more bandwidth than a regular, non-Redundancy-protected transmission. In order to avoid spending additional bandwidth when it is not needed, design of Redundancy **MUST** allow defining which DTP packets should be considered as Redundancy Packets. Currently we use a F flag from DTP Packet Header to indicate whether a packet is Redundancy-protected or not. The format of header will be described in Section 4.3 later.

The Redundancy Data generated in Redundancy module **MUST** be distinguished from application data payload. Redundancy Data should not be transferred to the application upon reception, they are indeed generated by and for the Redundancy Scheme used by the transport protocol. We use Redundancy Packet to transmit Redundancy Data Section 4.3.

There are multiple Redundancy Scheme candidates. During the handshake process, a scheme will be negotiated for the DTP session, just like encryption scheme negotiation. Currently DTP specifically chooses Reed-Solomon FEC Scheme as described in [arXiv_1809.04822].

3.5. Loss Detection and Congestion Control

This document reuses the congestion control module defined in QUIC [QUIC]. Congestion control module is responsible to send packets, collects ACK and do packet loss detection. Then it will put the lost data back to the retransmission queue of each block. Congestion control module is also responsible to monitor the network status and report the network condition such as bandwidth and RTT to scheduler.

4. Extension of QUIC

DTP is implemented as an extension of QUIC by **mapping QUIC stream to DTP block one to one**. In that way, DTP can reuse the QUIC stream cancellation mechanism to drop the stale block during transmission. And DTP can also utilize the max stream data size defined by QUIC to negotiate its max block size. Besides, the block id of DTP can also be mapped to QUIC stream id without breaking the QUIC stream id semantic.

DTP implements its block dropping mechanism by leveraging QUIC's stream cancellation function. DTP only defines the drop action on the **sender side** to cancel stale blocks. DTP leaves the decisions to the application layer on the receiver side to determine whether to accept an overdue block. However, because QUIC allows to cancel streams on both sides and DTP is an extension of QUIC, DTP **MAY** cancel the block from the receiver side. It requires mechanisms to measure each receiving block's importance and drop it.

DTP endpoints communicate by exchanging packets. And the payload of DTP packets, consists of a sequence of complete frames. As defined in [QUIC], each frame begins with a Frame Type, indicating its type, followed by additional type-dependent fields. Besides the many frame types defined in Section 12.4 of [QUIC], DTP introduces BLOCK_INFO Frame to support timeliness data transmission. And DTP also makes adjustment on QUIC ACK Frame. Another extension is introducing FEC packet to support FEC.

4.1. New Frame: BLOCK_INFO Frame

DTP adds two kinds of BLOCK_INFO frames (type=0x20, 0x21). Either of these frames **SHOULD** be attached in the front of each block to inform the scheduler of Block Size, Block Priority, and Block Deadline. These parameters can be used to do block scheduling. The BLOCK_INFO frame is as follows:

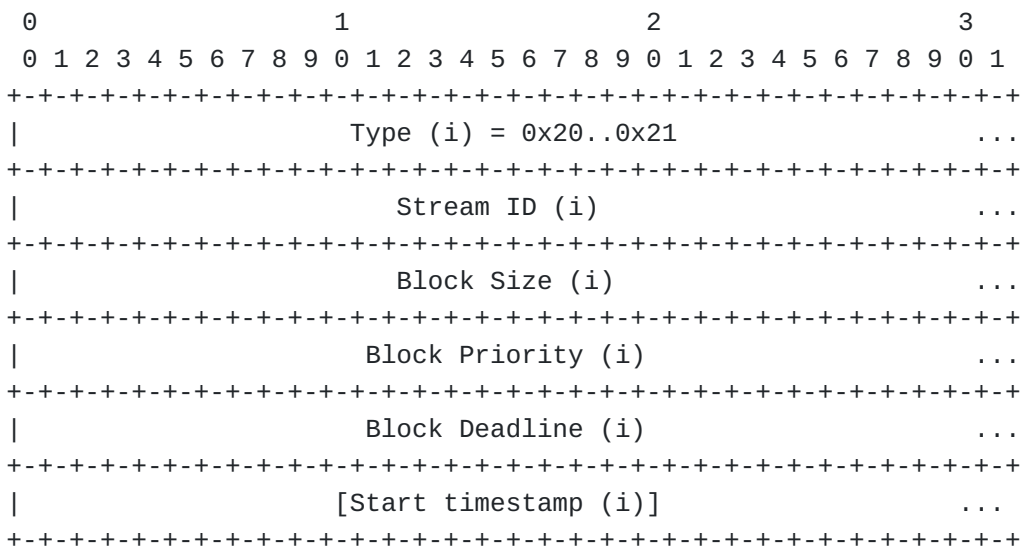

Figure 3: BLOCK_INFO Frame Format

*Stream ID: A variable-length integer indicating the stream ID of the stream.

*Block Size: A variable-length integer indicating the size of the block.

*Block Priority: A variable-length integer indicating the priority of the block.

*Block Deadline: A variable-length integer indicating the required transimission deadline. Dealine should be a duration in microseconds.

*Start timestamp: An optional parameter to inform the receiver about the starting time of this block. This parameter may be helpful when the receiver wants to use the deadline information. The timestamp parameter **SHOULD** be the same format as Unix timestamp. The sender and receiver **SHOULD** do clock synchronization if they use this parameter.

4.2. New Frame: Timestamped ACK Frame

DTP adds a new Timestamped ACK Frame, containing a timestamp to carry the timeliness information. The receiver sends Timestamped ACK Frame to inform the sender when a packet is received and processed. ACK mechanism of DTP is almost the same as QUIC. The format of Timestamped ACK frames is similar to the standard ACK Frames defined in section 19.3 of [QUIC] (As shown in Figure 4):

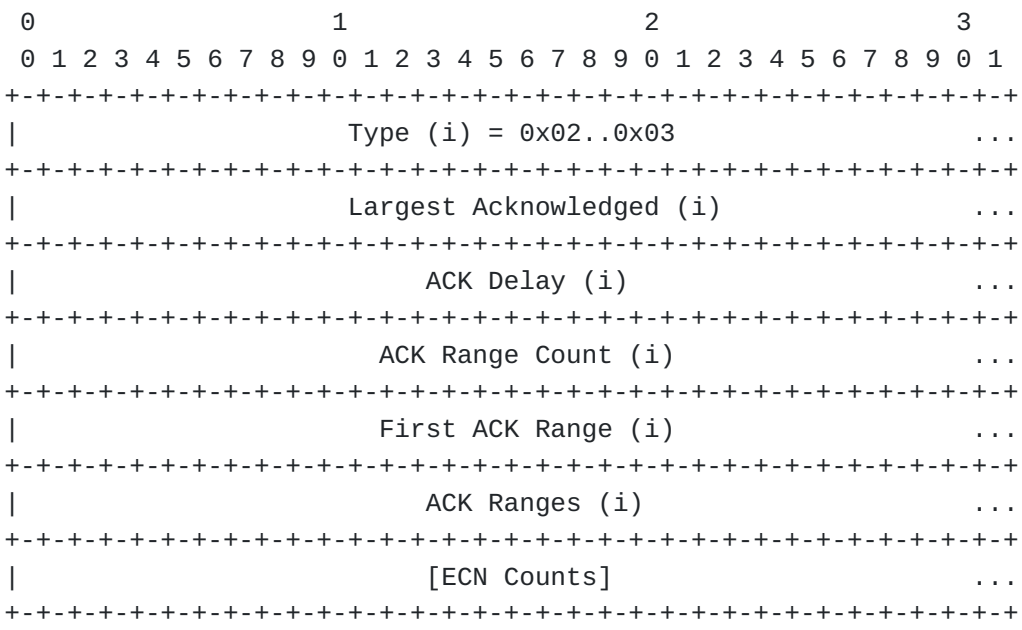

Figure 4: Standard QUIC ACK Frame Format

DTP appends a timestamp parameter after the original QUIC ACK Frame Format and defines the type of this new frame 0x22..0x23 (As shown in Figure 5). The timestamp parameter can be regarded as an optional parameter of the QUIC ACK Frame while using an extension frame type.

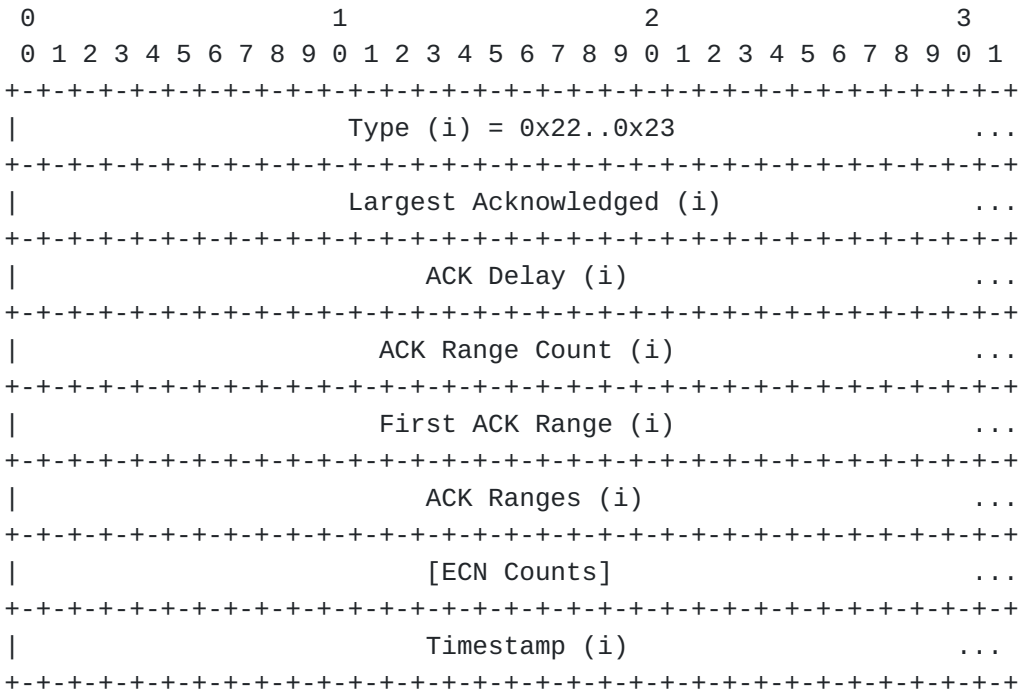

Figure 5: Timestamped ACK Frame Format

Using this timestamp parameter, we can calculate whether the prior blocks transmitted misses the deadline or not, and we can also calculate the block completion rate before the deadline. The timestamp parameter **SHOULD** be in the same format as Unix timestamp.

The Timestamped ACK is adequate to inform the sender about the timeliness information from the receiver side. To fully use the deadline information in the block, the sender and the receiver **SHOULD** do clock synchronization.

4.3. New Packet: Redundancy Packet

We use a F Flags in DTP Packet to distinguish which DTP packets is Redundancy-protected or not. Figure 6 shows the Redundancy Packet Format. If the flag is set, the Redundancy Group ID, m, n, index field is appended to the header. They are used by the Redundancy Scheme(Forward-Error-Correction) to identify the redundancy-protected data and communicate information about the encoding and decoding procedures to the receiver-side Redundancy Scheme.

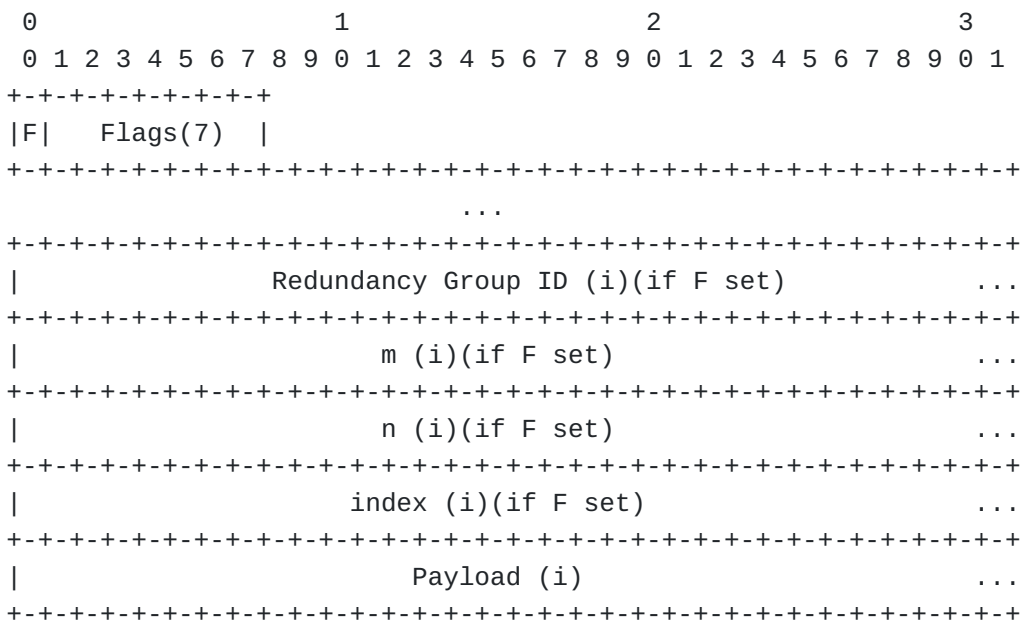

Figure 6: Redundancy Packet Format

*F: A flag indicating whether this DTP packets is FEC-protected or not.

*Redundancy Group ID: A variable-length integer indicating the id of the redundancy group which the packet belongs to.

*m: A variable-length integer indicating the number of original packets of the redundancy group.

*n: A variable-length integer indicating the number of redundancy packets of the redundancy group.

*index: A variable-length integer indicating the location of the packet inside the redundancy group.

*Payload: The payload of the Redundancy Packet, containing DTP Payload or Redundancy Data.

5. DTP Use Cases

5.1. Block Based Real Time Application

DTP can provide deliver-before-deadline service for Block Based Real Time Applications. Applications like real-time media and online multiplayer gaming have deadline requirements for their data transimission. These application also tend to generate and process the data in block fashion, for example, video/audio encoder produces the encoded streams as a series of block (I,B,P frame or GOP). And these real-time applications usually have multiple blocks (As shown in Figure 1) to be transferred simultaneously. DTP can optimize the data transmission of these applications by scheduling which block to be sent first. And Redundancy Module of DTP can reduce retransmission delay.

5.2. API of DTP

5.2.1. Data Transmission Functions

5.2.1.1. Send

Format: SEND(connection id, buffer address, byte count, block id, block deadline, block priority) -> byte count

The return value of SEND is the continuous bytes count which is successfully written. If the transport layer buffer is limited or the flow control limit of the block is reached, application needs to call SEND again.

Mandatory attributes:

*connection id - local connection name of an indicated connection.

*buffer address - the location where the block to be transmitted is stored.

*byte count - the size of the block data in number of bytes.

*block id - the identity of the block.

*block deadline - deadline of the block.

*block priority - priority of the block.

5.2.1.2. Update

Format: UPDATE(connection id, block id, block deadline, block priority) -> result

The UPDATE function is used to update the metadata of the block. The return value of UPDATE function indicates the success of the action. It will return success code if succeeds, and error code if fails.

Mandatory attributes:

*connection id - local connection name of an indicated connection.

*block id - the identity of the block.

*block deadline - new deadline of the block.

*block priority - new priority of the block.

5.2.1.3. Retreat

Format: RETREAT(connection id, block id) -> result

The RETREAT function is used to cancel the block. The return value of RETREAT function indicates the success of the action. It will return success code if succeeds, and error code if fails.

Mandatory attributes:

*connection id - local connection name of an indicated connection.

*block id - the identity of the block.

5.2.1.4. Receive

Format: RECV(connection id, buffer address, byte count, [,block id]) -> byte count, fin flag, [,block id]

The RECV function shall read the first block in-queue into the buffer specified, if there is one available. The return value of RECV is the number of continuous bytes which is successfully read, and fin flag to indicate the ending of the block. If the block is cancelled, the RECV function will return error code BLOCK_CANCELLED. It will also returns the block id on which it receives if application does not specify it.

If the block size specified in the RECV function is smaller than the size of the receiving block, then the block will be partial copied(indicated by the fin flag). Next time RECV function is called, the remaining block will be copied, and the id will be the same. This fragmentation will give extra burden to applications. To avoid the fragmentation, sender and receiver can negotiate a max block size when handshaking.

Mandatory attributes:

*connection id - local connection name of an indicated connection.

*buffer address - the location where the block received is stored.

*byte count - the size of the block data in number of bytes.

Optional attributes:

*block id - to indicate which block to receive the data on.

5.2.2. Feedback Functions

5.2.2.1. on_dropped

Format: ON_DROPPED(connection id) -> block id, deadline, priority, goodbytes

The ON_DROPPED function is called when a block is dropped. The metadata of the dropped block such as block id, deadline, priority is attached. The number of bytes delivered before its deadline(goodbytes) is returned.

Mandatory attributes:

*connection id - local connection name of an indicated connection.

5.2.2.2. on_delivered

Format: ON_DELIVERED(connection id) -> block id, deadline, priority, delta, goodbytes

The ON_DELIVERED function is called when a block is delivered. The metadata of the delivered block such as block id, deadline, priority is attached. The number of bytes delivered before its deadline (goodbytes) and the difference between the block completion time and the deadline (delta) are returned.

Mandatory attributes:

*connection id - local connection name of an indicated connection.

All these functions mentioned above are running in asynchronous mode. An application can use various event driven framework to call those functions.

5.3. Collaborate with upper layer protocols

Application protocol on top of DTP may benefit from the block info and detail metric of the transport layer. DTP **MAY** expose the block information to the receiver side application and the status of the congestion control and buffer status to both sender side and receiver side application. This information will enable multiple DTP relay node working together to improve the deadline-delivery performance end-to-end.

6. Design Considerations

6.1. Clock Synchronization

The fundamental design of DTP relies on precise clock synchronization. The block scheduler requires high clock precision to accurately perform block canceling functions and efficient scheduling. Timestamped ACKs also necessitate high clock precision to enable the server and client to utilize deadline information effectively. However, achieving high precision clock synchronization across the web poses challenges. Further discussions are required to explore how to best utilize the deadline information in such circumstances.

6.2. Block Dependency

Video streams often exhibit decoding dependencies among their frames. To address this, it would be beneficial to include block dependencies as critical metadata in the block info. Our basic design involves adding an integer field to the block info frame, indicating the stream id on which the current block depends. This enhancement may facilitate efficient block processing and playback, ensuring that frames are correctly ordered and decoded based on their dependencies.

6.3. Automatic Block Info

DTP receives block priorities and block deadlines from the send and update API. However, determining appropriate values for these parameters can be challenging for applications. Even in cases where applications, such as RTC publishers, aim for transport delays below 100ms, they may not get the ideal transport result by setting the block deadline parameter to 100ms. To address this, it might be necessary to devise an automatic method that can recognize the application's requirements and assign rational parameter values accordingly. Implementing such an automatic mechanism can streamline

the configuration process for applications, freeing developers from the burden of manually fine-tuning parameters and ensuring optimal data transmission with minimal human intervention.

7. Security Considerations

See the security considerations in [QUIC] and [QUIC-TLS]; the block-based data of DTP shares the same security properties as the data transmitted within a QUIC connection.

8. IANA Considerations

This document has no IANA actions.

9. Normative References

[arXiv_1809.04822] Michel, F., Coninck, Q., and O. Bonaventure, "Adding Forward Erasure Correction to QUIC", September 2018, <https://arxiv.xilesou.top/pdf/1809.04822.pdf>.

[QUIC] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[QUIC-TLS] Thomson, M., Ed. and S. Turner, Ed., "Using TLS to Secure QUIC", RFC 9001, DOI 10.17487/RFC9001, May 2021, <https://www.rfc-editor.org/rfc/rfc9001>.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/rfc/rfc768>.

[RFC0793] Postel, J., "Transmission Control Protocol", RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/rfc/rfc793>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

Acknowledgments

We sincerely thank Z. Liu and J. Zhang for contributing to the DTP project. They provided a lot of advice and revisions to the draft

and actively helped advance the relevant progress of DTP standardization.

Authors' Addresses

Yong Cui
Tsinghua University
30 Shuangqing Rd
Beijing
China

Email: cuiyong@tsinghua.edu.cn

Chuan Ma
Tsinghua University
30 Shuangqing Rd
Beijing
China

Email: mc21@mails.tsinghua.edu.cn

Hang Shi
Huawei

Email: shihang9@huawei.com

Kai Zheng
Huawei

Email: kai.zheng@huawei.com

Wei Wang
Huawei

Email: wangwei375@huawei.com