QUIC
Internet-Draft
Intended status: Informational
Expires: May 20, 2020

H. Shi
Y. Cui
Z. Liu
Tsinghua University

November 17, 2019

Deadline-aware Transport Protocol
draft-shi-quic-dtp-00

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.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 2
  1.1. Conventions 2
2. Motivation 2
3. Design of DTP 4
  3.1. Abstraction 4
  3.2. Architecture of DTP 4
  3.3. Deadline-aware Scheduler 6
  3.4. Deadline-aware Redundancy 6
  3.5. Loss Detection and Congestion Control 7
4. Extension of QUIC 7
5. API of DTP 7
6. IANA Considerations 11
7. Security Considerations 11
8. Normative References 11
Authors' Addresses 11

1. Introduction

Many emerging applications have the deadline requirement for their data transmission. However, current transport layer protocol 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

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, NOT RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in [RFC2119].

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.

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 blocks 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, since application is not supposed 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 building blocks 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 to facilitate the scheduling decision. Application can attach metadata along with the data block, those metadata include:

o 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.

o 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.

3.2. Architecture of DTP

The sender side architecture is shown in Figure 1:

```
                       +-------------+
                       |             |
                       | Application |
                       |             |
                       +-------------+
                              |
                              |
                              V
   +-----------------------------------------------------+
   |        0               1                    n       |
   | +----+---+---+  +----+---+---+       +----+---+---+ |
   | |Data| D | P |  |Data| D | P |  ...  |Data| D | P | |<--+ Lost
   | +----+---+---+  +----+---+---+       +----+---+---+ |   | Packet
   |        D: Deadline         P: Priority              |   |
   +-----------------------------------------------------+   |
                              |                              |
                              |                              |
                              V                              |
                       +-------------+                       |
                       |             |                       | Bandwidth
                       |  Scheduler  |<----------------------+ &
                       |             |                       | RTT
                       +-------------+                       |
                              |                              |
                              |                              |
                              V                              |
                       +-------------+                       |
                       |             |                       |
                       | Redundancy  |<----------------------+ Loss
                       | Encoder     |                       | Rate
                       +-------------+                       |
                              |                              |
                              |                              |
                              V                              |
                       +-------------+                       |
                       |             |                       |
                       | Congestion  +-----------------------+
                       | Control     |
                       +-------------+
```

Figure 1: The Architecture of DTP

In receiver side, the transport layer will receive data and reassemble the block. The process is symmetric with the sender side. It first goes through packet parsing and redundancy processing module. Transport layer also keeps track of the deadline of each block. When receiver calls RECV function (Refer to [Section 5]), the

transport layer returns the received in-ordered data to the application.

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's 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.

3.4. 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. The

redundancy module can help mitigate that problem by generating the redundancy packets to avoid retransmission. Since only retransmission of tail packets of the block will increase the block completion time so the redundancy is only applied to tail packets of each block. The tail packets is defined within the Bandwidth-Delay Product range of the block. And blocks with higher priority also get more redundancy.

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.

5. API of DTP

DTP extends the send socket API to let application attach metadata along with the data block, and the API of DTP is structured as follows:

A) Data transmission functions

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.

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.

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.

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.

B) Feedback functions

Status

Format: STATS(connection id, block id) -> byte count

The STATS function is used to query the deadline delivery result. The application uses STATS to query the bytes delivered before the deadline to receiver of each block. The information can be used to adjust the block sending rate of each priority. For example, if the application finds that the lowest priority block always get dropped due to the limited bandwidth, the application can stop generating the block to save the computation power. Combined the status of each priority, the application can also get the overall network capacity to facilitate the rate adaptation algorithm.

Mandatory attributes:

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

* block id - the identity of the block.

Block Completion Time (BCT)

Format: QUERY_BCT(connection id, block id) -> block completion time

After receiving the block, application can query the block completion time using QUERY_BCT. This can also facilitate the rate or deadline adaptation of application. For example, if the base RTT of the network is bigger than deadline, then all blocks will miss the deadline. In this case, application may choose to relax its deadline.

Mandatory attributes:

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

* block id - the identity of the block.

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

6. IANA Considerations

This document has no actions for IANA.

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. Normative References

[QUIC] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-20 (work in progress), April 2019.

[QUIC-TLS] Thomson, M. and S. Turner, "Using TLS to Secure QUIC", draft-ietf-quic-tls-20 (work in progress), April 2019.

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

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/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/info/rfc2119>.

Authors' Addresses

Hang Shi
Tsinghua University
30 Shuangqing Rd
Beijing
China

Email: shi-h15@mails.tsinghua.edu.cn

Yong Cui
Tsinghua University
30 Shuangqing Rd
Beijing
China

Email: cuiyong@tsinghua.edu.cn

Zhiwen Liu
Tsinghua University
30 Shuangqing Rd
Beijing
China

Email: liu-zw16@mails.tsinghua.edu.cn