

Congestion Control Working Group
Internet-Draft
Intended status: Informational
Expires: 3 September 2026

P. Ageneau
G. Armitage
S. Danahy
Netflix
2 March 2026

Network Delivery Time Control
draft-ageneau-ccwg-ndtc-01

Abstract

This document describes Network Delivery Time Control (NDTC), a rate adaptation algorithm for real-time video streaming suited for interactive applications like cloud gaming. NDTC leverages the Frame Dithering Available Capacity Estimation (FDACE) heuristic, which estimates available path capacity without inducing congestion. The algorithm dynamically adjusts frame sizes and transmission times to ensure timely delivery, while also responding to conventional congestion signals.

About This Document

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

The latest revision of this draft can be found at <https://paullouisageneau.github.io/draft-ageneau-ccwg-ndtc/draft-ageneau-ccwg-ndtc.html>. Status information for this document may be found at <https://datatracker.ietf.org/doc/draft-ageneau-ccwg-ndtc/>.

Discussion of this document takes place on the Congestion Control Working Group Working Group mailing list (<mailto:ccwg@ietf.org>), which is archived at <https://mailarchive.ietf.org/arch/browse/ccwg/>. Subscribe at <https://www.ietf.org/mailman/listinfo/ccwg/>.

Source for this draft and an issue tracker can be found at <https://github.com/paullouisageneau/draft-ageneau-ccwg-ndtc>.

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 3 September 2026.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	3
2. Conventions and Definitions	4
3. General Principle	4
4. Algorithm Overview	5
4.1. Illustrating using RTP/RTCP	5
4.2. Architecture Considerations	6
4.2.1. Sender-side Agent	6
4.2.2. Receiver-side Agent	6
4.2.3. Timing and Timestamping	7
4.3. Frame Dithering Available Capacity Estimation (FDACE)	7
4.4. Target Frame Size from Available Capacity	10
4.5. Combined AIMD Congestion Control	10
4.6. Encoder Target Frame Size	12
4.7. Adaptive Frame Pacer	13
5. Implementation Details	15
5.1. Variables and Parameters	15
5.2. Frame Size Calculation	18
5.3. Loss Detection	18
6. Fairness Considerations	19
7. Active Queue Management Considerations	19
8. Security Considerations	19
9. IANA Considerations	20
10. Normative References	20
Appendix A. Dual-variable EWMA Process Implementation	21
Appendix B. Estimate Implementation	22

Appendix C. AIMD Logic Implementation	22
Authors' Addresses	23

1. Introduction

This document specifies implementation details and considerations for Network Delivery Time Control (NDTC) [NDTC-LCN], a rate adaptation and congestion control algorithm for real-time, interactive video flows which adapts the target frame size to fit within the available path capacity, proactively minimizing self-induced congestion. NDTC also reacts to traditional packet loss signals and is compatible with networks that implement the Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service.

NDTC's characteristics make it ideal for high-bitrate, real-time, interactive applications:

- * It does not self-induce congestion, keeping queueing delays and packet loss to a minimum,
- * Ramp up is fast with no risk of overshoot, even at high bitrates,
- * The encoded video is not required to closely follow the target bitrate, and there is no need to send filler data.

These properties make it particularly suited for interactive cloud gaming applications, and (potentially) for bitrate ladder rung selection in low-latency video applications.

At a very high level, when NDTC is applied to cloud gaming it continuously runs the following loop:

- * Pace each video frame's packets at a rate between 1x and 2x (on average) our current estimate of available path capacity, in order to probe the network path for additional capacity while minimising burst load on network bottlenecks.
- * Set the target frame size of the video encoder to a fraction of the available capacity estimate, thus keeping frame delivery latencies low.

Each frame's paced packet train is a probe to test and update our estimate of available capacity, and thus continuously update our upper bound on the next encoded video frame's size.

2. 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.

3. General Principle

A key requirement for real-time video is ensuring each frame arrives on time to be presented. Consequently, NDTC focuses on timely frame delivery rather than attempting to maximise throughput. NDTC aims to ensure each frame reception duration is shorter than a frame period, providing sufficient headroom to ensure high probability of each frame arriving in time. In practice, the control loop of the algorithm stabilizes the frame reception duration by dynamically setting the target frame size to a fraction of the estimated path capacity.

NDTC introduces and relies on Frame Dithering Available Capacity Estimation (FDACE), a heuristic that paces packetized frames over a dithered send duration and estimates available path capacity from the relationship between send durations and observed receive durations. Importantly, capacity estimation is performed without injecting additional traffic (beyond the video stream we wish to deliver) or filling the path bottleneck to capacity. We then ensure that the flow uses only a fraction of the measured capacity to prevent risks of self-induced congestion.

This process is combined with an AIMD (Additive Increase Multiplicative Decrease) congestion control process for compatibility with both packet loss congestion signals and ECN (Explicit Congestion Notification) [RFC9331] for the L4S (Low Latency, Low Loss, and Scalable Throughput) Internet Service [RFC9330].

For the rest of this document we describe NDTC based on the following mental model:

- * The server is encoding video on-the-fly at a frame rate $1/T_{FRAME}$ (where T_{FRAME} is the frame period), while capping each frame to a video frame size target $TARGET$.
- * The server forms a path capacity probe by pacing the frame's constituent packets out over target send duration T_{SEND} .

- * The path's estimated available capacity AVAILABLE is continuously updated using each frame's actual send and receive durations (SEND and RECV respectively).
- * TARGET is recalculated whenever AVAILABLE changes to ensure the next RECV is stabilized around target reception duration TREC.V.
- * Additionally, TARGET is capped by a combined AIMD process reacting to congestion signals like packet loss and ECN.

To properly probe the path, TSEND MUST be a fraction of TREC.V, and to minimise on-path congestion, TREC.V MUST be a fraction of TFRAME. We RECOMMEND that $TREC.V = 0.6 * TFRAME$ and $TSEND = 0.5 * TREC.V$. (For example, sending at 30fps would result in $TFRAME = 33ms$, $TREC.V = 20ms$ and $TSEND = 10ms$.)

Note: NDTC focuses on uncovering enough available capacity to meet the application's needs, it is not trying to instantly discover all the available capacity on a path. Specifically, choosing $TSEND = 0.5 * TREC.V$ limits FDACE to probing up to $2.0 * AVAILABLE$ in any single iteration of the FDACE heuristic.

NDTC initializes TARGET to INIT_TARGET, and whenever TARGET is recalculated the result is constrained to $MIN_TARGET \leq TARGET \leq MAX_TARGET$. Actual values for the INIT_TARGET, MIN_TARGET and MAX_TARGET parameters are application-specific.

4. Algorithm Overview

This section details implementation of the whole NDTC feedback loop using variable and parameter names summarized in section Section 5.1 (rather than the more mathematically-oriented names used in NDTC's conference paper [NDTC-LCN]).

4.1. Illustrating using RTP/RTCP

Although NDTC can be implemented for various real-time video transport protocols, this document illustrates the use of NDTC to support applications using RTP/RTCP [RFC3550].

In particular, NDTC operates on a single video stream and it is frame-oriented. References to a "frame" in the rest of this document mean a sequence of consecutive RTP packets having the same RTP timestamp and ending with a packet whose RTP header Mark bit is set.

Defining a frame when NDTC is used for non-RTP flows is currently outside the scope of this document.

4.2. Architecture Considerations

NDTC can be implemented both sender-side or receiver-side depending on the application's needs.

In the following discussion, the "agent" refers to the entity running the NDTC algorithm. A core requirement to execute FDACE is that, for every frame, the agent learns the precise durations over which a given frame was sent (SEND) and received (RECV). Recommendations for transmitting this timing information between senders and receivers are discussed in subsequent sections.

If the agent is the sender, the algorithm runs on every frame feedback. If the agent is the receiver, the algorithm runs as soon as a frame is received and sends the updated target frame size (TARGET) to the server.

4.2.1. Sender-side Agent

Sender-side NDTC can be instantiated by having receivers pass RECV back to the sender inside existing RTCP-based Feedback for Congestion Control [RFC8888] or Transport-wide Congestion Control (TWCC) [draft-holmer-rmcat-transport-wide-cc-extensions] reports. Alternatively, an implementor might introduce application-specific RTCP feedback to make messages smaller and achieve better precision.

The receiver is also required to report which packets have not been received (as described in Section 5.3) and SHOULD also reflect Explicit Congestion Notification (ECN) markings of received packets (if support for L4S is desired).

In this scenario, the agent has local control of the underlying packet pacer and video encoder.

4.2.2. Receiver-side Agent

Receiver-side NDTC requires the receiver to regularly pass back updates to TARGET (to properly control the video encoder) and SLOPE (section Section 4.3) to properly control the packet pacer. This may be achieved with e.g. RTCP messages like Receiver Estimated Maximum Bitrate (REMB) [draft-alvestrand-rmcat-remb].

Note: if the receiver agent cannot pass back SLOPE information, the sender's pacing heuristic SHOULD assume SLOPE = 1.

4.2.3. Timing and Timestamping

The precision with which SEND and RECV are measured and reported MUST be 1ms or better, because the durations are typically around a few milliseconds only, and a lower precision would not be sufficient to obtain a consistent estimate, even by averaging multiple samples. We RECOMMEND a precision of 0.1ms.

If the agent is implemented sender-side, we assume the agent derives SEND from local knowledge (e.g. timestamps generated low in the networking stack), and learns RECV from reports passed back by the receiver.

If the agent is implemented receiver-side, we assume the agent derives RECV from local knowledge (e.g. kernel timestamps from the underlying network interface) and learns SEND via in-band messages from the sender. For instance, the sender can write the effective send time on each packet just before it goes out using the [abs-send-time] RTP extension.

4.3. Frame Dithering Available Capacity Estimation (FDACE)

FDACE calculates an estimated available capacity AVAILABLE and a slope SLOPE given send and receive durations for a frame.

Frame size LENGTH is calculated by summing up packet payload sizes for the frame according to Section 5.2. Therefore AVAILABLE will actually refer to application-layer capacity expressed in terms of video bitrate.

If the frame consists of a single packet, if LENGTH is strictly less than MIN_TARGET, or if the frame suffered packet loss in the sense of Section 5.3, the agent does not run FDACE. Instead, it retains the TARGET and SLOPE values from the most recent frame for which FDACE ran and immediately jumps to the AIMD congestion control described in Section 4.5.

If the frame was packetized as 2 packets or more and suffered no packet loss, the agent first calculates the normalized send duration NSEND and the normalized reception duration NRECV for the frame by respectively dividing frame send duration SEND and frame receive duration RECV by frame size LENGTH.

$$\text{NSEND} = \text{SEND} / \text{LENGTH}$$
$$\text{NRECV} = \text{RECV} / \text{LENGTH}$$

Send duration SEND corresponds to the elapsed time between sending the first packet and sending the last packet of the frame (i.e. the one with the RTP marker). SEND MUST be the actual duration over which the pacer sent the frame, not the calculated target duration from Section 4.7. Packet send times SHOULD be measured as close to socket send calls as possible.

Receive duration RECV corresponds to the elapsed time between reception of the first packet and reception of the last packet of the frame (i.e. the one with the RTP marker). Note there is no need for sender and receiver clock synchronization. In practice, packet reception times SHOULD be measured using kernel timestamps for enhanced precision, as the application might take significant time to receive packets in userspace. The agent SHOULD cap RECV to a small multiple of the frame period in order to mitigate outliers caused by transient adverse network events while still allowing the value to grow large enough to capture bursts of cross-traffic and reflect sudden path capacity reductions. The RECOMMENDED maximum value for RECV is $3 \cdot T_{FRAME}$.

A dual-variable EWMA process is updated with the new sample (NSEND, NRECV). The process calculates the averages AVG_NSEND, AVG_NRECV, the variances VAR_NSEND and VAR_NRECV, and the covariance COVAR. The RECOMMENDED weight is $LAMBDA = 0.04$. A suitable implementation of the dual-variable EWMA process is detailed in Appendix A.

The agent now derives the linear regression parameters from the output of the EWMA process. The slope SLOPE and y-intercept INTERCEPT of the linear equation $NRECV = SLOPE \cdot NSEND + INTERCEPT$ are calculated as follows:

```
if (VAR_NSEND > 0.0 && COVAR > 0.0) {
    SLOPE = min(COVAR / VAR_NSEND, 1.0)
} else {
    SLOPE = 0.0
}
INTERCEPT = max(AVG_NRECV - SLOPE * AVG_NSEND, 0.0)
```

SLOPE reflects how pacing frames changes their reception as they go through the path bottleneck and intercept more or less cross-traffic. The relationship is locally linear because we observe normalized times and not rates, and network path components behave linearly in terms of processing time per byte to a good approximation.

If the path is not constraining, for instance during ramp-up, receive duration is always equal to send duration, and $SLOPE = 1$, which can be interpreted as the path capacity looking infinite.

If there is no visible cross-traffic and the flow is alone at bottleneck, the slope SLOPE is 0, the send rate does not matter as we always receive at bottleneck rate. If cross-traffic takes most of the bottleneck capacity, SLOPE is close to 1. More generally, we can calculate that in a single FIFO bottleneck scenario with constant-rate cross traffic, SLOPE is the fraction of the bottleneck capacity occupied by cross-traffic. However, this statement does not hold true in all scenarios. FDACE does not require such a simple scenario and performs correctly in more complex network setups, for instance with more advanced AQM (Active Queue Management) disciplines (see Section 7).

The y-intercept INTERCEPT reflects the normalized receive duration that we would measure if the frame was sent as an unpaced burst, reflecting total path capacity.

The agent extrapolates ESTIMATE, the normalized reception duration that we would measure if the frame was sent at receive rate, reflecting available capacity on the path. It corresponds to the ordinate of intersection with the identity line $NRECV = NSEND$.

In order to be robust when SLOPE is close or equal to 1, ESTIMATE is approximated with successive iterations from AVG_NRECV and converging to the intersection. The RECOMMENDED number of iterations is $ITERATIONS = 3$. If $ITERATIONS$ is too low, NDTC will overestimate the available capacity in the presence of cross-traffic. If $ITERATIONS$ is higher, NDTC yields more easily against elastic cross-traffic.

```
ESTIMATE = AVG_NRECV
for (i = 0; i < ITERATIONS; i++) {
    ESTIMATE = SLOPE * ESTIMATE + INTERCEPT
}
```

This means that when the flow is not constrained and $SLOPE = 1$, $ESTIMATE = AVG_NRECV$, which is a conservative estimate for the capacity.

For increased simplicity and performance in CPU-constrained environments, the loop MAY be unrolled as noted in Appendix B.

The agent SHOULD add a conservative margin MARGIN to ESTIMATE, increasing with the linear regression error. The idea is that when the model fits poorly, the agent should add a higher margin as a safety buffer. The RECOMMENDED formula to calculate MARGIN is to multiply a constant KMARGIN by the standard deviation of NRECV and $1-R^2$ where R^2 is the coefficient of determination representing the proportion of variation in NRECV explained by the linear regression. The RECOMMENDED value for KMARGIN is 0.25. A value too close to 0

makes the algorithm react slower to abrupt changes. A value too high for KMARGIN makes the algorithm significantly underestimate the capacity in the presence of bursty cross traffic.

```
if (VAR_NSEND > 0.0 && VAR_NRECV > 0.0) {  
    R2 = COVAR^2 / (VAR_NSEND * VAR_NRECV)  
    MARGIN = KMARGIN * sqrt(VAR_NRECV) * (1-R2)  
} else {  
    MARGIN = 0.0  
}
```

Available capacity AVAILABLE is calculated by taking the reciprocal of the sum ESTIMATE + MARGIN:

```
AVAILABLE = 1.0 / (ESTIMATE + MARGIN)
```

4.4. Target Frame Size from Available Capacity

Before FDACE and the combined congestion control run for the first time, TARGET is initialized to INIT_TARGET and SLOPE is initialized to 1.0.

Each time FDACE runs and updates the available capacity, the agent now derives the new target frame size TARGET from TRECV and AVAILABLE and caps it to MAX_TARGET.

```
TARGET = min(TRECV * AVAILABLE, MAX_TARGET)
```

The agent MUST cap TARGET to MAX_TARGET even if the encoder has its own internal maximum frame size to prevent a runaway feedback loop in case the path capacity is very large. Without capping, NDTC would continue increasing the target frame size until it matches the available path capacity even if the encoded video can't follow it, pacing over shorter and shorter durations in the process. Frames would be sent as very short bursts, increasing the risk of packet loss. MAX_TARGET SHOULD be a sensible frame size that the video content can realistically reach.

4.5. Combined AIMD Congestion Control

Even if FDACE can successfully run on its own most of the time, it doesn't react to conventional indications of network congestion, in particular packet loss, whereas reacting to packet loss is essential in the presence of a policer on the path. Therefore, NDTC combines an AIMD (Additive Increase Multiplicative Decrease) process with FDACE to also adjust the frame size and send duration in response to conventional congestion feedback. It allows the algorithm to seamlessly transition between capacity estimation and congestion

signals, following the path evolution.

The AIMD process computes a congestion target frame size CTARGET and a pseudo-slope CSLOPE to respectively cap TARGET and SLOPE, in order to limit the effective pacer rate. See Section 4.7 for details about how the pacer adapts the sending duration according to the slope.

- * When CTARGET < TARGET, the flow is congestion-limited, therefore TARGET is set to CTARGET and CSLOPE is set to 0.0 so the pacer sends at rate CTARGET / TREC.V.
- * When CTARGET >= TARGET, the flow is not congestion limited, therefore TARGET is unchanged and CSLOPE is set so that the pacer send duration results in a send rate around CTARGET / TREC.V.
- * When CTARGET reaches TARGET * TREC.V / TSEND, CSLOPE reaches 1.0 and the pacer reaches its minimum send duration and maximum send rate around TARGET/TREC.V.

Therefore, the maximum congestion frame size for the AIMD logic is defined to correspond to the maximum send rate:

$$C_{MAX} = TARGET * TREC.V / TSEND$$

For each transmitted frame, the agent identifies whether packets of the frame were lost as described in Section 5.3, and runs the AIMD logic to update a congestion frame size CSIZE. CSIZE SHOULD be initialized to MAX_TARGET as in most cases FDACE should be self-sufficient.

If one or more packets were lost, the agent performs a decrease by capping CSIZE to CMAX and multiplying it by BETA. If no packets were lost and CSIZE is less than CMAX, the agent increases CSIZE by adding ALPHA, then caps the result to CMAX. The RECOMMENDED values are ALPHA=40 bytes and BETA=0.7 to make increase and decrease relatively conservative and minimize QoE reduction.

The AIMD logic SHOULD NOT feature a slow start phase and both decrease and increase SHOULD be suppressed for one round-trip after a loss event. In order to support L4S [RFC9330], the agent SHOULD also react similarly to Prague congestion control [draft-briscoe-iccrp-prague-congestion-control], which requires supporting ECN [RFC9331], and if the agent is the sender, it also requires compatible feedback like [RFC8888]. If the last ECN decrease is more recent than the last packet loss event, the increase SHOULD be performed by adding a larger EALPHA instead of ALPHA for the flow to compete more fairly with other flows. The RECOMMENDED value is EALPHA=400 bytes.

The RECOMMENDED algorithm is described in Appendix C.

If the agent is the sender, it SHOULD also perform the decrease in case it does not get any feedback for a significant duration after sending a frame. In any case, the sender SHOULD implement a circuit breaker to stop sending after it stops receiving feedback for a significant duration, following the recommendations in [RFC8083].

The congestion frame size CTARGET is calculated as the minimum of CSIZE and CMAX:

$$CTARGET = \min(CSIZE, CMAX)$$

Then, CSLOPE is calculated such that the average pacing rate is CTARGET/TRECV. This is obtained by solving for CSLOPE in the pacer equation in Section 4.7 such that when LENGTH = TARGET, the send duration results in an effective send rate of CTARGET/TRECV:

$$CSLOPE = \max(1 - (TSEND/TRECV) * (CMAX/CTARGET), 0) / (1 - TSEND/TRECV)$$

4.6. Encoder Target Frame Size

The agent caps the target frame size TARGET and the slope SLOPE with CTARGET and CSLOPE respectively to take into account the most conservative estimate:

$$\begin{aligned} TARGET &= \min(TARGET, CTARGET) \\ SLOPE &= \min(SLOPE, CSLOPE) \end{aligned}$$

The agent MUST then floor TARGET to MIN_TARGET:

$$TARGET = \max(TARGET, MIN_TARGET)$$

Flooring to MIN_TARGET is required to ensure that the whole feedback loop does not get stuck producing 1-packet frames, which do not allow the agent to run FDACE and update TARGET. The RECOMMENDED value for MIN_TARGET is 2000 bytes as a frame of such a size will be packetized into 2 packets of significant size given a typical path MTU.

The sender SHOULD set the encoder target frame size to TARGET as soon as possible (i.e. the encoder target bitrate is set to TARGET/TFRAME).

4.7. Adaptive Frame Pacer

When a new video frame is generated at the sender, it is first packetized into RTP packets. The exact packetization process depends on video codec. The sender SHOULD first pad the frame to MIN_TARGET with codec-specific bitstream filler data if it is smaller. A frame of size MIN_TARGET or larger MUST be packetized in at least 2 packets. The sender SHOULD also attempt to packetize the frame into packets of similar sizes. The sender MAY add RTP padding as specified in [RFC3550] to help make packet sizes similar.

The sender now calculates the desired send duration SEND for the frame.

When not at bottleneck, the send duration is around target send duration TSEND. To ensure a positive feedback loop, TSEND MUST be strictly less than TREC, the RECOMMENDED value is $TSEND = 0.5 * TREC$ and $TREC = 0.3 * TFRAME$.

The sender first calculates the base pacing duration PACE by adding dithering of amplitude DELTA to TSEND and interpolating with TREC according to SLOPE. The RECOMMENDED value for DELTA is $0.5 * TSEND$. With `rand()` a pseudo-random value in $[-1, 1]$, PACE is calculated as:

$$PACE = SLOPE * (TSEND + rand() * DELTA) + (1 - SLOPE) * TREC$$

Adapting the duration according to SLOPE reduces how aggressively NDTC paces packets as the flow gets closer to filling the entire bottleneck. It makes the feedback loop run closer to the desired operating point and minimises transient queueing for each frame. It also helps FDACE by probing locally in non-linear scenarios and stabilizes competition with other NDTC flows.

The sender then scales the pacing duration proportionally to LENGTH/TARGET, where the frame size LENGTH is calculated according to Section 5.2. This helps sustaining the send rate when the encoder produces frames that don't match the target frame size. For instance, the encoder may chronically underproduce because the video is not dynamic or complex enough, and it may temporarily overproduce on scene change. Eventually, SEND is capped at the frame period TFRAME.

$$SEND = \min(PACE * LENGTH/TARGET, TFRAME)$$

The sender also calculates a frame alignment delay as follows to reduce frame jitter. The delay is calculated to compensate the send duration variation so the last packets of frames are received at regular frame period intervals:

$$\text{DELAY} = \text{SLOPE} * \max(\text{PACE} + \text{SLOPE} * \text{DELTA} - \text{SEND}, 0)$$

The packets of the frame are sent after DELAY over the duration SEND, meaning that if the frame is available at t , the first packet is sent at $t + \text{DELAY}$ and the last packet (i.e. the one with the RTP marker) should be sent at $t + \text{DELAY} + \text{SEND}$. Other packets are spread evenly over the interval.

To achieve this, the sender SHOULD wait for duration DELAY, then for each packet p , send it, then wait for an interval $\text{SEND} * \text{SIZE}(p) / \text{LENGTH}$ where $\text{SIZE}(p)$ is the payload size for packet p . The pacer SHOULD match send times with an absolute precision of at least 1ms. It MUST prevent cumulative errors from actual wait durations being different from expected durations.

If ECN is not supported, the pacer does not need to precisely pace packets besides the first and last one, and it MAY instead group packets in small bursts to prevent waking up too often.

In any case, the agent MUST use the actually achieved send duration, not the calculated pacing duration, when running FDACE for the frame.

If FEC is enabled, the sender SHOULD NOT send FEC packets during duration SEND and SHOULD instead pace FEC packets after it, spreading them so that the interval between packets is calculated similarly to the interval for video packets.

If the sender supports retransmissions, the sender SHOULD handle retransmitted packets separately. They SHOULD be paced and sent in parallel to video frames. If the application features an audio stream, the sender SHOULD handle audio packets separately. They SHOULD be sent immediately after they are generated, without adding a delay. This means retransmissions and audio packets can be sent between two video packets. From the point of view of FDACE, they will be accounted for as cross-traffic sharing the same path.

The pacer MUST handle the corner case where packets from the previous frame are still waiting to be sent when the new frame is available, for instance because the new frame is early. In this case, it is crucial to ensure that the previous frame is fully sent before the new frame is sent. Therefore if packets from the previous frame were to be sent after the first packet of the new frame, the sender MUST reschedule the leftover packets to send them before the first packet of the new frame.

5. Implementation Details

5.1. Variables and Parameters

The following table summarizes parameters used by the algorithm:

Parameter	Description	Unit	Recommended
MIN_TARGET	minimum target frame size	bytes	2000
MAX_TARGET	maximum target frame size	bytes	-
INIT_TARGET	initial target frame size	bytes	$\leq \text{MAX_TARGET}/2$
TRECV	target receive duration	seconds	$0.6 * \text{TFRAME}$
TSEND	target send duration	seconds	$0.5 * \text{TRECV}$
ITERATIONS	extrapolation iterations	-	3
ALPHA	congestion additive increase	bytes	40
EALPHA	congestion additive increase for ECN	bytes	400
BETA	congestion multiplicative decrease	-	0.7
DELTA	send dithering amplitude	seconds	$0.5 * \text{TSEND}$
LAMBDA	regression EWMA weight	-	0.04
KMARGIN	estimation margin constant	-	0.25
TFRAME	video frame period	seconds	application-specific

Table 1

The following table summarizes variables used by the algorithm:

Variable	Description	Unit
AVAILABLE	estimated available capacity	bytes/second
DELAY	pacing alignment delay	seconds
TARGET	target frame size from FDACE	bytes
CTARGET	target frame size from AIMD	bytes
LENGTH	actual frame size	bytes
CMAX	AIMD maximum frame size	bytes
CSIZE	AIMD frame size	bytes
PACE	base pacing duration	seconds
SEND	send duration for the frame	seconds
RECV	receive duration for the frame	seconds
SLOPE	linear regression slope (a)	-
INTERCEPT	linear regression y-intercept (b)	seconds/byte
ESTIMATE	extrapolated estimate	seconds/byte
MARGIN	estimation margin	seconds/byte
CSLOPE	pseudo-slope from AIMD	-
NSEND	normalized send duration	seconds/byte
NRECV	normalized receive duration	seconds/byte
AVG_NSEND	average of norm. send duration	seconds/byte
AVG_NRECV	average of norm. receive duration	seconds/byte
VAR_NSEND	variance of norm. send duration	(seconds/byte)^2
VAR_NRECV	variance of norm. receive duration	(seconds/byte)^2
COVAR	covariance of norm. send and receive durations	(seconds/byte)^2

Table 2

5.2. Frame Size Calculation

The frame size LENGTH is used for normalizing the send and receive durations in FDACE and calculating the pacing duration. LENGTH is calculated as the sum of RTP payloads corresponding to the frame (i.e. sharing the same timestamp). FEC and retransmitted packets MUST NOT be counted in LENGTH.

Since frame size is calculated from payload sizes only, FDACE will estimate an application-layer capacity, and TARGET can be set directly as encoder target. Implementors MAY choose to take into account RTP headers, and optionally UDP/IP headers. In that case, capacity will more closely reflect network capacity, however target frame size SHOULD be decreased by the typical header overhead before being set as encoder target.

If the frame consists of two packets or more, send and receive durations do not account for one of the extreme packets. When sending, since the pacer waits after sending each packet, the sender SHOULD sum up payload sizes except the last one. When running FDACE, the agent does not know how the path exactly handles packets, so it can't know for sure which payload to exclude at reception. Therefore, in the context of FDACE, the agent SHOULD calculate LENGTH as the sum of all payloads of the frame minus the average size of the first and last one. If the frame consists of a single packet, LENGTH is the size of the packet.

Note that if the frame is packetized into packets of similar sizes as recommended, which packet to exclude does not matter since the payload sizes are the same.

5.3. Loss Detection

Packet loss is taken into account to decide if FDACE should be run for a frame and to trigger AIMD decrease. Lost packets are detected by tracking discontinuities in received sequence numbers.

The receiver SHOULD reorder packets for the currently received frame (i.e. timestamp) to avoid misclassifying reordered packets as lost. As soon as the first packet of the next frame (i.e. timestamp) is received, missing packets for the previous frame MUST be reported as lost. The receiver MUST consider a packet as lost in the context of this algorithm even if it can be recovered with Forward Error Correction or retransmission. Therefore, a frame triggering packet loss reaction may still be successfully transmitted.

6. Fairness Considerations

As NDTC does not attempt to maximize throughput, it also does not attempt to achieve throughput fairness in general. Targeting a reception duration means that NDTC tends to yield to capacity seeking-flows, but it does not get starved.

Thanks to its design, NDTC never takes more capacity than the combined AIMD congestion control logic allows, so it is guaranteed not to starve traditional capacity-seeking flows.

7. Active Queue Management Considerations

As interactive flows in general, NDTC flows greatly benefit from AQM (Active Queue Management) disciplines.

In the case of NDTC, flow queueing, for instance in FQ-CoDel [RFC8290] or Cake [CAKE], particularly helps FDACE as cross-traffic packets are interleaved in each frame, providing stability in addition to fairness. As NDTC adapts the send duration to pace close to available capacity, Cake's DRR++ scheme additionally allows NDTC to have priority over bulk traffic.

NDTC can also take advantage of L4S bottlenecks if the AIMD logic behaves as a Prague congestion controller like suggested in Appendix C, providing fairness with TCP Prague and traditional TCP flows. An essential aspect is that NDTC can seamlessly transition between L4S and standard bottlenecks as the network path evolves.

8. Security Considerations

An attacker who could insert, edit, or drop messages from the connection could cause NDTC to underutilize the path capacity or overshoot it, causing network congestion. The RTP flow and RTCP feedback should be protected from message injection and modification using SRTP.

In a more sophisticated attack, an attacker could selectively delay video packets in order to manipulate timings and make NDTC incorrectly estimate available path capacity. However, the combined congestion control should prevent NDTC from causing network congestion in that scenario.

9. IANA Considerations

This document has no IANA actions.

10. Normative References

[abs-send-time]

"RTP Header Extension for Absolute Sender Time", n.d.,
<<http://www.webrtc.org/experiments/rtp-hdrext/abs-send-time>>.

[CAKE]

"Cake - Common Applications Kept Enhanced", n.d.,
<<https://www.bufferbloat.net/projects/codel/wiki/Cake>>.

[draft-alvestrand-rmcat-remb]

Alvestrand, H., "RTCP message for Receiver Estimated Maximum Bitrate", 21 October 2013,
<<https://datatracker.ietf.org/doc/html/draft-alvestrand-rmcat-remb-03>>.

[draft-briscoe-iccrp-prague-congestion-control]

Schepper, K. D., Tilman, O., Briscoe, B., and V. Goel, "Prague Congestion Control", n.d.,
<<https://datatracker.ietf.org/doc/html/draft-briscoe-iccrp-prague-congestion-control-04>>.

[draft-holmer-rmcat-transport-wide-cc-extensions]

Holmer, S., Flodman, M., and E. Sprang, "RTP Extensions for Transport-wide Congestion Control", 19 October 2015,
<<https://datatracker.ietf.org/doc/html/draft-holmer-rmcat-transport-wide-cc-extensions-01>>.

[NDTC-LCN] Ageneau, P.-L. and G. Armitage, "Network Delivery Time Control: a Novel Approach to Rate Adaptation for Low-Latency Video Flows", IEEE 50th Conference on Local Computer Networks (LCN), 13 October 2025,
<<https://ieeexplore.ieee.org/document/11146354>>.

[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>>.

- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <<https://www.rfc-editor.org/rfc/rfc3550>>.
- [RFC8083] Perkins, C. and V. Singh, "Multimedia Congestion Control: Circuit Breakers for Unicast RTP Sessions", RFC 8083, DOI 10.17487/RFC8083, March 2017, <<https://www.rfc-editor.org/rfc/rfc8083>>.
- [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>>.
- [RFC8290] Hoeiland-Joergensen, T., McKenney, P., Taht, D., Gettys, J., and E. Dumazet, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm", RFC 8290, DOI 10.17487/RFC8290, January 2018, <<https://www.rfc-editor.org/rfc/rfc8290>>.
- [RFC8888] Sarker, Z., Perkins, C., Singh, V., and M. Ramalho, "RTP Control Protocol (RTCP) Feedback for Congestion Control", RFC 8888, DOI 10.17487/RFC8888, January 2021, <<https://www.rfc-editor.org/rfc/rfc8888>>.
- [RFC9330] Briscoe, B., Ed., De Schepper, K., Bagnulo, M., and G. White, "Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service: Architecture", RFC 9330, DOI 10.17487/RFC9330, January 2023, <<https://www.rfc-editor.org/rfc/rfc9330>>.
- [RFC9331] De Schepper, K. and B. Briscoe, Ed., "The Explicit Congestion Notification (ECN) Protocol for Low Latency, Low Loss, and Scalable Throughput (L4S)", RFC 9331, DOI 10.17487/RFC9331, January 2023, <<https://www.rfc-editor.org/rfc/rfc9331>>.

Appendix A. Dual-variable EWMA Process Implementation

All variables are initialized at zero. When receiving a new sample (NSEND, NRECV), the EWMA process SHOULD be updated as follow:

```
COUNT = COUNT + 1
WEIGHT = max(LAMBDA, 1.0/COUNT)
DELTA_NSEND = NSEND - AVG_NSEND
DELTA_NRECV = NRECV - AVG_NRECV
AVG_NSEND += WEIGHT * DELTA_NSEND
AVG_NRECV += WEIGHT * DELTA_NRECV
VAR_NSEND = (1 - WEIGHT) * (VAR_NSEND + WEIGHT * DELTA_NSEND^2)
VAR_NRECV = (1 - WEIGHT) * (VAR_NRECV + WEIGHT * DELTA_NRECV^2)
COVAR = (1 - WEIGHT) * (COVAR + WEIGHT * DELTA_NSEND * DELTA_NRECV)
```

COUNT is also initialized at zero and the weight $1.0/\text{COUNT}$ is used at the beginning to ensure initial values get equal weights and prevent the very first value from having an unreasonably large effect on the average.

Appendix B. Estimate Implementation

ESTIMATE is calculated step-by-step with ITERATIONS iterations, and the RECOMMENDED value for ITERATIONS is 3. In a CPU-constrained environment, for instance if NDTC runs for a very large number of parallel sessions, the loop MAY therefore be unrolled, making ESTIMATE linear according to AVG_NRECV:

$$\text{ESTIMATE} = \text{SLOPE}^3 * \text{AVG_NRECV} + (\text{SLOPE}^2 + \text{SLOPE} + 1) * \text{INTERCEPT}$$

Appendix C. AIMD Logic Implementation

On frame feedback, the agent SHOULD run the following AIMD logic to update the congestion frame size CSIZE:

```
ecn_gain = 1.0 / 16.0
ecn_fraction = float(ecn_marking_count) / float(packet_count)
ecn_average += ecn_gain * (ecn_fraction - ecn_average)

// Multiplicative decrease
if (last_decrease_time > first_packet_send_time) {
    // Suppress decrease for one round-trip
} else if (packet_lost_count > 0) {
    // Loss decrease
    CSIZE = min(CSIZE, CMAX) * BETA
    last_decrease_time = now()
} else if (last_ecn_decrease_time > first_packet_send_time) {
    // Suppress ECN decrease for one round-trip
} else if (ecn_marking_count > 0) {
    // ECN decrease
    CSIZE = min(CSIZE, CMAX) * (1.0 - ecn_average * (1.0 - BETA))
    last_ecn_decrease_time = now()
}

// Additive increase (suppressed after loss but not ECN decrease)
if (last_decrease_time <= first_packet_send_time && CSIZE < CMAX) {
    if (last_ecn_decrease_time <= last_decrease_time) {
        CSIZE = min(CSIZE + ALPHA, CMAX)
    } else {
        CSIZE = min(CSIZE + EALPHA * (1.0 - ecn_fraction), CMAX)
    }
}
```

This algorithm uses logic close to a Prague congestion controller [draft-briscoe-iccrp-prague-congestion-control], it also uses the recommended ECN gain 1/16. For a new flow, `ecn_average` SHOULD be initialized to 1.0.

If ECN is not supported, the AIMD logic MAY be simplified to only the multiplicative decrease on packet loss (suppressed for one round-trip) and the additive increase otherwise.

Authors' Addresses

Paul-Louis Ageneau
Netflix
Email: pageneau@netflix.com

Grenville Armitage
Netflix
Email: garmitage@netflix.com

Internet-Draft

NDTC

March 2026

Scott Danahy
Netflix
Email: sdanahy@netflix.com