Adaptive Receiver-Side Scheduling for Smooth Interactive Delivery*

Michael Luby
BitRipple, Inc.
luby@bitripple.com

Abstract

Interactive applications such as cloud gaming, XR streaming, and real-time inference depend on data objects arriving at a steady cadence. In practice, network delay variation and recovery dynamics at the receiver distort this cadence even when transports deliver all packets correctly, which produces visible jitter, stalls, and unstable playback.

We present a lightweight receiver-side scheduling approach that regularizes release timing after recovery. The scheduler maintains an adaptive estimate of effective path delay and adjusts release times asymmetrically, responding quickly to late arrivals and only gradually to early ones. This upper-envelope behavior keeps release aligned with recent delay peaks and maintains smooth playback with minimal added latency. The scheduler runs entirely on the receiver clock and requires no feedback or synchronization.

As a concrete example, we integrate receiver-side scheduling into the BitRipple Tunnel (BRT) overlay [1], an application-layer software system that forwards traffic without altering the underlying transport protocol. Within BRT, the scheduler functions as an independent module that regulates delivery timing for forwarded objects.

Evaluating BRT with receiver-side scheduling on a cloud-gaming workload shows that the scheduler removes virtually all large jitter excursions and yields tightly clustered release intervals that improve visible smoothness. Broader latency improvements arise from the behavior of the full BRT overlay. Receiver-side scheduling can also be integrated modularly into transport stacks such as TCP, QUIC, WebRTC, UDP, or RTP, which are natural deployment points for future work.

1 Introduction

Interactive applications place strict timing requirements on the receiver. Cloud gaming, XR streaming, real-time telepresence, and distributed inference all depend on a continuous flow of objects, such as video frames, that must be delivered at a stable cadence. Variation in packet arrival times causes the recovery time of each object to fluctuate, so even when every packet is delivered correctly the receiver observes irregular recovery times. Small fluctuations in these recovery times translate directly into visible jitter, buffer oscillation, and head-of-line stalls. Because recovery often occurs in bursts or with uneven gaps due to changing network delay, immediately releasing objects upon recovery exposes this variability to the application and disrupts interactive performance.

Sender-side techniques help but do not remove the problem. Pacing can smooth departures when path conditions are stable, but it cannot correct variability introduced by queueing, reordering, or recovery dynamics that only the receiver observes. Larger playback buffers can hide jitter, yet they increase end-to-end latency and reduce interactivity. Traditional jitter buffers were designed for short audio packets and use fixed or slowly adaptive depth rules that are poorly matched to the variability

*This material is based upon work supported in part by the National Science Foundation under Award 2212574.

patterns seen in ultra-low-latency interactive applications such as cloud gaming and teleoperation, whether each object is a single packet or reconstructed from many.

This paper introduces a receiver-side scheduling approach that smooths object release timing by operating at the point where recovery-time variability becomes visible. The scheduler observes when each object was generated and when it is recovered, and uses this information to release objects with a stable cadence. The scheduler does not attempt to estimate path delay. It instead infers the required timing offset implicitly by comparing sender timestamps with local recovery times, rising quickly when recovery is late and falling only gradually when recovery is early. This upper-envelope behavior keeps the schedule aligned with recent delay peaks and ensures that most objects complete recovery before their scheduled release. The result is smooth playback with minimal added latency. The scheduler runs entirely on the receiver clock and requires no feedback or synchronization.

Receiver-side scheduling includes additional mechanisms useful in practical systems. Quantized scheduling locks release times to increments aligned with the application cadence, which suppresses micro-level jitter. A bounded in-order guard preserves sequencing when recovery completes out of order while preventing unbounded waiting for delayed predecessors. These refinements rely only on local timing information and do not alter the sender or underlying transport.

The approach integrates cleanly with overlay systems such as the BitRipple Tunnel (BRT), a deployed software system designed for immersive and interactive traffic [1]. BRT operates at the application layer between cooperating endpoints and forwards objects without altering the application's transport protocol. In this environment, receiver-side scheduling runs as a separate module that regulates when recovered objects are delivered to the application.

The design is transport agnostic. Receiver-side scheduling can be integrated after reassembly or recovery in systems based on TCP, QUIC, WebRTC, RTP, UDP, or similar protocols. It is equally compatible with receive paths that reconstruct larger objects from packets before presenting them to the application.

Contributions.

- A receiver-side scheduling approach that stabilizes release cadence by adapting release timing to the observed recovery pattern, including asymmetric offset adjustment, quantized scheduling, and bounded in-order release.
- An integration of this approach into the BitRipple Tunnel (BRT) overlay system, which shows that receiver-side timing control can be added as a modular component without modifying the application's transport protocol.
- A real-world evaluation of BRT with receiver-side scheduling, using a cloud-gaming workload, that shows substantial reductions in jitter, significantly tighter release intervals, and materially improved percentile latency consistency.

2 Related Work

Receiver-side buffering and playout control have been widely studied across real-time media and streaming systems.

Early adaptive playout algorithms for packet audio estimated network delay and dynamically adjusted the receiver playout point to smooth jitter [11]. Later work integrated forward error correction with delay adaptation [12] and refined jitter-buffer control for VoIP environments to balance delay against late-loss probability [10, 2]. These designs typically reset or retune delay during talkspurts and assume independent, small packets rather than object recovery.

Within the RTP/RTCP framework, inter-arrival jitter measurement and receiver-side buffering are standardized in the RTP specification [13] and further refined for transmission-time offsets [15]. Most RTP implementations maintain a fixed or slowly adaptive jitter buffer whose depth is chosen heuristically. In practice this means using a preset value, often 50–100 ms, or a simple moving-average rule such as

$$\text{buffer} = \text{mean delay} + k \times \text{std deviation},$$

with k empirically selected, commonly between 1 and 2. Some systems enlarge the buffer after several late packets and then shrink it gradually when conditions stabilize, while others ship codec-specific presets determined through laboratory testing and subjective playback evaluation. These methods rely on heuristics tuned by measurement and experience rather than on an explicit control model or analytic optimization.

Industrial low-latency streaming protocols expose similar receiver buffer parameters. The Secure Reliable Transport (SRT) protocol defines a latency configuration that directly sets the receiver buffer depth and consequently the minimum end-to-end delay [6, 5]. The IETF Internet-Draft for SRT describes buffering behavior but does not specify a receiver-side scheduling algorithm beyond this adjustable delay window [14]. In practice, SRT and related contribution protocols rely on static configuration and are sensitive to network variability.

In HTTP adaptive streaming, buffer management is primarily used for rate adaptation rather than post-recovery scheduling. Buffer-based controllers select segment bitrates based on playback buffer occupancy [7], and work such as FESTIVE analyzes the fairness and stability of such adaptive bitrate schemes [8]. These mechanisms regulate transmission rate and encoding level rather than inter-release timing at the receiver.

The present work differs in focusing solely on scheduling after recovery, independent of transport or sender pacing. The adaptive asymmetric update, quantized hysteresis, and bounded in-order guard operate entirely on the receiver clock and can coexist with RTP, SRT, QUIC, or coded overlays. This perspective complements classic playout-delay control by introducing a general receiver-side scheduling layer between recovery and application delivery.

3 Model and Problem Statement

We consider a sender that generates objects indexed by n with sender timestamp S_n and a receiver that recovers them at times A_n on its local clock. The task is to choose release times T_n that satisfy three goals:

(i) preserve order or permit only bounded overtake,

(ii) keep the transmission-to-release delay small and controlled,

(iii) match the inter-release intervals to the sender's generation intervals as closely as possible.

Clocks are not synchronized. Receiver-side scheduling uses sender timestamps as metadata and otherwise operates entirely on the receiver clock. In many systems the sender timestamp equals the object generation time, which makes cadence matching meaningful for interactive playback and buffer stability.

Design goals. We target four concrete properties that guide the controller:

(G1) one-to-one correspondence between sent and delivered objects,

(G2) preservation of the original order except for bounded, controlled overtake,

(G3) equality of inter-object spacing at sender and receiver up to small bounded error,

(G4) minimal additional latency beyond what is needed for smooth delivery.

4 Adaptive Receiver-Side Scheduling

This section formalizes the Adaptive Delay Control (ADC) algorithm. The receiver maintains an offset D that maps sender timestamps onto the receiver's time axis. For any object with sender timestamp S_n, the expression $S_n + D$ represents a projected release time on the receiver clock. The scheduler then compares this projected value against the recovery time A_n to determine when the object should be released. The offset D is therefore a continuously adjusted translation between the sender and receiver clocks, chosen so that the projected release time $S_n + D$ lies at or just beyond the recovery time for most objects. The release time is always at least the larger of the recovery time and the projected value, and it keeps the scheduled times close to the upper envelope of the observed recovery times. The controller updates D asymmetrically, reacting quickly to late arrivals while reducing the offset gradually when arrivals are early.

4.1 State, Variables, and Initialization

Let D denote the adaptive mapping offset. For the first recovered object with (S_0, A_0), initialize

$$D \leftarrow A_0 - S_0,$$

so that the initial projected release time $S_0 + D$ coincides with its recovery time A_0.

To prevent drift during idle periods, if no recoveries occur for longer than a timeout T_{idle}, the controller re-anchors the offset when activity resumes by setting

$$D \leftarrow A_n - S_n$$

for the first object n recovered after the idle gap. This reinitialization restores alignment between sender timestamps and the receiver's clock after a prolonged pause. The controller does not reinitialize for any object that is recovered within T_{idle} of its predecessor.

For each subsequent recovered object n, define the deviation

$$X_n = A_n - (S_n + D),$$

which is the difference, in receiver time, between the recovery time and the projected release time implied by the current offset. Positive deviations indicate that the object recovered later than the projected schedule; negative deviations indicate recovery earlier than projected.

4.2 Asymmetric Update

To stabilize scheduling while preserving responsiveness to delay spikes, ADC applies an asymmetric update to D. The update uses exponent parameters $0 \le \rho_u \le 1 \le \rho_\ell$, damping parameters $\lambda_u > 0$ and $\lambda_\ell > 0$, and a clipping parameter $U > 0$. A typical choice for these parameters is $\rho_u = 0.5$, $\rho_\ell = 1.5$, $\lambda_u = 0.05$, $\lambda_\ell = 0.03$, and $U = 1$ second.

Given X_n, ADC computes an adjustment Y_n and then updates D as follows:

$$\text{if } X_n > 0: \quad Y_n = \left(\frac{\min\{X_n, U\}}{U}\right)^{\rho_u} \cdot U, \qquad D \leftarrow D + \lambda_u \cdot Y_n,$$

$$\text{if } X_n < 0: \quad Y_n = -\left(\frac{\min\{|X_n|, U\}}{U}\right)^{\rho_\ell} \cdot U, \qquad D \leftarrow D + \lambda_\ell \cdot Y_n.$$

The exponent parameters are chosen so that the algorithm reacts more strongly to late recoveries than to early ones. This effect arises primarily from the asymmetric shaping of Y_n by ρ_u and ρ_ℓ:

- **Exponent asymmetry.** Since $0 \le \rho_u \le 1$, the function $a \mapsto a^{\rho_u}$ is concave on $[0, U]$. For any late deviation $X_n = a \in (0, U]$,
$$Y_n = \left(\frac{a}{U}\right)^{\rho_u} \cdot U \ge a,$$
so the change in D overshoots the observed lateness before damping by λ_u. This overshoot allows the projected release timeline to move above current delay peaks.

  Since $\rho_\ell \ge 1$, the function $a \mapsto a^{\rho_\ell}$ is convex on $[0, U]$. For any early deviation $X_n = -a$ with $a \in (0, U]$,
$$|Y_n| = \left(\frac{a}{U}\right)^{\rho_\ell} \cdot U \le a,$$
so the change in D undershoots the amount by which recovery is early. This yields gentler downward motion into delay valleys.

- **Damping parameters.** The damping parameters λ_u and λ_ℓ scale these overshoot and undershoot adjustments before they are added to D. They control how aggressively the offset responds to the shaped deviations and can be chosen independently. In practice, choosing λ_u and λ_ℓ on the order of a few percent of Y_n produces gradual but responsive adaptation.

Together, the concave late-response shaping and convex early-response shaping, combined with damping, shift the projected release timeline forward rapidly when recovery lags ($X_n > 0$), reducing the risk of late release, while allowing it to drift downward more cautiously when recovery is early. This keeps the projected release times aligned with the delay peaks rather than the midpoint of recovery variability.

We optionally apply a bound $\delta > 0$ on waiting through
$$D \leftarrow \min\big(D,\, A_n - S_n + \delta\big),$$
which limits the projected release time $S_n + D$ to exceed the recovery time A_n by at most δ.

Why asymmetry helps. Recovery patterns naturally contain peaks and valleys. If reactions to early and late recoveries were symmetric, the offset D would track the midpoint of these fluctuations and would routinely fall below the true delay peaks, which increases the risk of late release and playback stalls. As shown in Section 13, the symmetric linear case $\rho_u = \rho_\ell = 1$ cannot maintain the projected timeline at or above the upper envelope of a fluctuating recovery pattern; instead it converges to a value strictly between the high and low delays. The exponent asymmetry described above directly addresses this limitation.

When $X_n > 0$ is small (object recovered slightly late), the concave shaping with $\rho_u < 1$ produces an adjustment Y_n whose magnitude exceeds X_n before damping, so the projected release timeline is pushed upward by more than the observed lateness. This overshoot makes it easier for subsequent objects to recover before their scheduled release. When $X_n < 0$ is small (object recovered slightly early), the convex shaping with $\rho_\ell > 1$ produces an adjustment whose magnitude is less than $|X_n|$, so the projected timeline moves downward only slightly. The combination stays near the upper envelope of the recovery pattern and avoids deep excursions into delay valleys.

A further benefit comes from the stability of inter-release spacing. When the projected release time of an object is after its recovery time, the release interval matches the original inter-send

interval provided that D remains nearly constant. Because the asymmetric update rules make early recoveries more common than late ones, and the early adjustments are undershooting, D changes very little from one object to the next even under varying network conditions. As a result, the inter-release times at the receiver stay close to the inter-send times at the sender, which supports smooth playback.

Optional neutral band. An optional neutral band parameter $J \geq 0$ can be used to suppress updates for modest early recoveries and to retain a small safety margin in the release rule. When $J > 0$ is enabled, the early-update condition and adjustment are modified to

$$\text{if } X_n \leq -J: \quad Y_n = -\left(\frac{\min\{|X_n + J|, U\}}{U}\right)^{\rho_\ell} \cdot U, \qquad D \leftarrow D + \lambda_\ell \cdot Y_n,$$

so that there is no update when $X_n \in (-J, 0]$. In this regime, the projected release time is already close enough to the recovery time that additional downward adjustment is unnecessary. In some deployments and in the evaluation below, J is set to zero and the core ADC behavior described above dominates. The neutral band parameter J, when used, suppresses small downward corrections and preserves this behavior while absorbing noise.

4.3 Release Scheduling

After updating D, the scheduler assigns the release time. In the core ADC form without a neutral band,

$$T_n = \max\{A_n,\, S_n + D\}.$$

All quantities are expressed on the receiver clock. This rule ensures that:

- an object is scheduled to be released after it has been recovered ($T_n \geq A_n$),
- the release cadence follows the sender's timing through the projected mapping $S_n + D$,
- inter-release spacing remains close to the inter-send spacing as long as D varies slowly.

When a neutral band $J > 0$ is enabled, the scheduler includes a small safety margin and uses

$$T_n = \max\{A_n,\, S_n + D + J\},$$

which places most recoveries shortly before their release times and maintains the envelope-tracking behavior described above.

5 Illustrative Behavior

Figure 1 compares raw recovery timing against the smoothed schedule generated by ADC. The envelope-following behavior is visible: when recovery rises above the schedule, the schedule rises quickly; in flat regions it decays only slowly, which limits added delay while keeping most arrivals ahead of their release deadlines.

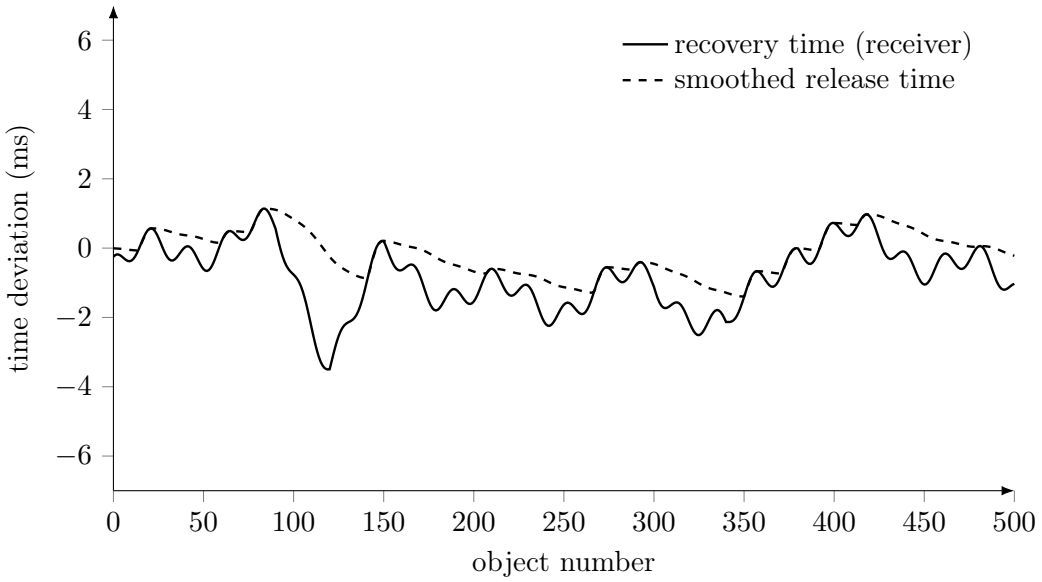

Figure 1: Recovery times versus smoothed release times.

6 System View

7 System View

Figure 2 shows where receiver-side scheduling fits in the receive pipeline. Packets from one or more network paths feed into a recovery module that reconstructs each object and produces its sender timestamp S_n together with its recovery time A_n. The scheduling module runs immediately after recovery and uses these (S_n, A_n) pairs to determine a release time R_n for each object. This placement ensures that the scheduler operates on the timing actually experienced by the receiver while remaining independent of the underlying transport protocol.

Because it observes only sender timestamps and local recovery times, the same module can follow TCP reassembly, QUIC datagram delivery, WebRTC frame construction, RTP depacketization, or object-based coded overlays such as BRT. The scheduler neither modifies transport behavior nor requires feedback to the sender; it simply regulates when recovered objects are presented to the application interface. This separation allows receiver-side scheduling to be added as a lightweight component in existing media and control pipelines.

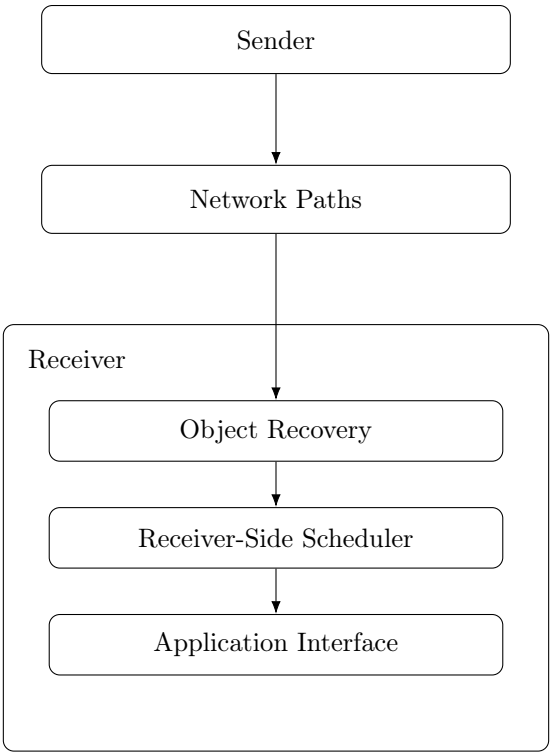

Figure 2: Receiver-side scheduling sits after recovery and before release to the application.

8 Enhanced ADC

Receiver-side scheduling in its basic ADC form already stabilizes release timing by maintaining an adaptive offset that tracks the upper envelope of observed recovery times. In practical deployments, however, several additional refinements improve cadence stability, sequencing behavior, and responsiveness to real network dynamics. These refinements operate entirely on the receiver clock, require no sender changes, and preserve the core control logic of ADC.

This section introduces three enhancements that build on the adaptive offset mechanism. Quantized scheduling regularizes small fluctuations by following the adaptive offset through discrete steps that align with the application cadence. The bounded in-order guard preserves sequencing when recovery completes out of order while avoiding unbounded head-of-line waiting. Finally, automatic parameter adaptation allows envelope scales, clamp limits, and guard windows to track prevailing RTT and recovery characteristics so that the scheduler remains well tuned across a wide range of path and workload conditions.

Each enhancement is optional and independent. Together they provide a practical extension of the basic ADC mechanism suitable for real-time applications such as cloud gaming, XR streaming, and interactive media pipelines.

8.1 Quantized Scheduling

ADC removes coarse jitter but may leave small object-to-object variations in D. Quantized ADC (QADC) introduces a quantized offset E that follows D in steps of size $\gamma > 0$, where γ is a configurable quantization step parameter. The result is a perceptually smooth cadence with bounded tracking error.

Midpoint hysteresis. Initialize

$$D \leftarrow A_0 - S_0, \qquad E \leftarrow D + \frac{\gamma}{2}.$$

Hold E while $D \in [E - \gamma, E]$. If D exits the band, re-anchor

$$E \leftarrow D + \frac{\gamma}{2},$$

which preserves $E \geq D$ and $0 \leq E - D < \gamma$. Release with

$$T_n = \max\{A_n,\, S_n + E + J\}.$$

Choosing γ on the order of the application cadence, such as a video frame interval, suppresses micro-stutter while keeping QADC agile.

Properties. The invariants $E \geq D$ and $0 \leq E - D < \gamma$ bound added latency and enforce discrete, predictable adjustments. Re-anchoring requires a finite movement in D, which prevents rapid toggling due to noise. Because re-anchoring sets $E \leftarrow D + \gamma/2$, the quantized offset E changes only when D moves by at least $\gamma/2$. Small fluctuations in D therefore leave E unchanged, so the inter-release cadence shifts only when the underlying offset changes by a meaningful amount. Figure 3 illustrates how $E(t)$ tracks the smoothed offset $D(t)$ with a hysteresis band of width γ and only re-anchors when $D(t)$ crosses the band boundary.

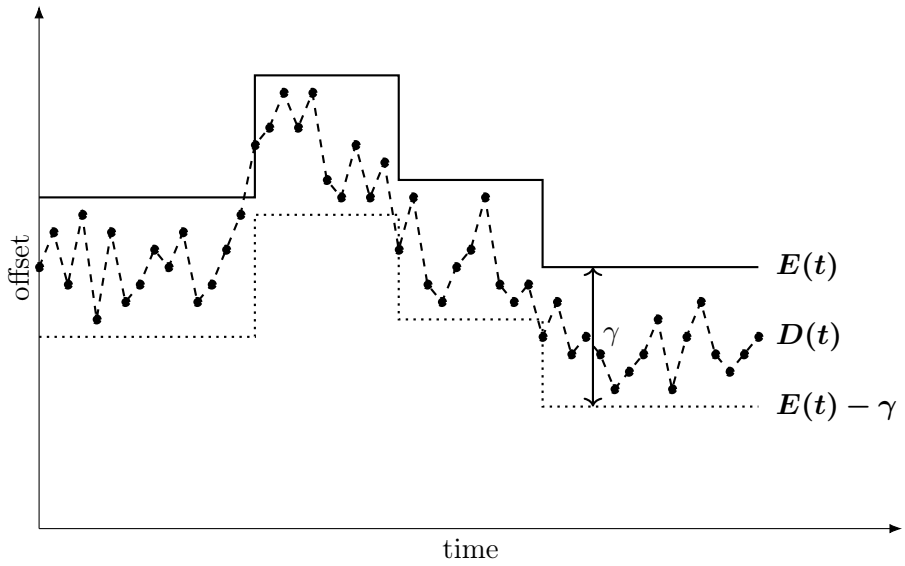

Figure 3: Quantized scheduling with midpoint hysteresis.

8.2 Bounded In-Order Guard

Some transports and applications assume ordered delivery. QADC with guard (QADC-G) adds a guard interval $G \geq 0$, a configurable guard-window parameter, to honor this assumption without creating unbounded head-of-line blocking.

Rule. With candidate T_n from ADC or QADC:

- if $n-1$ is released by T_n, release n at T_n,
- if $n-1$ releases in $[T_n,\, T_n+G)$, release n immediately after $n-1$,
- otherwise release n at T_n+G.

This rule preserves order when feasible while bounding additional waiting if a predecessor is delayed. Selecting G near a high percentile of observed reordering covers most cases without material latency cost. Figure 4 shows the guard window of length G around T_n, indicating when object n waits for its predecessor and when it releases at T_n+G if the predecessor remains missing.

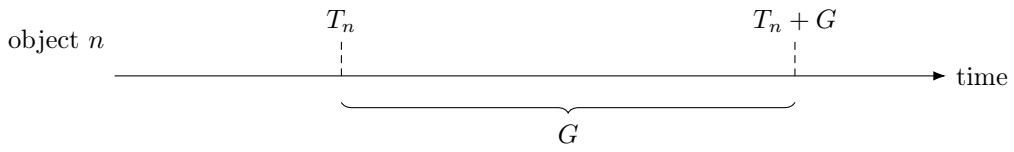

Figure 4: Bounded in-order guard window.

8.3 Automatic Parameter Adaptation

Several parameters in ADC, QADC, and QADC-G reflect the scale of variability seen at the receiver and need not remain fixed. In many systems the receiver has access to auxiliary measurements such as RTT (e.g., from QUIC, TCP, WebRTC feedback, or tunnel-level probing). These measurements operate on the receiver clock and provide a meaningful physical time scale, unlike the offset D, whose value depends on unrelated sender–receiver clock alignment.

This section describes how envelope, clamp, and guard parameters can be driven by RTT and by statistics of *inter-recovery intervals*, which are fully defined on the receiver clock and do not rely on sender timestamps.

Envelope scale from RTT. The clipping parameter U governs how aggressively the algorithm responds to large bursts in recovery timing. Because U represents a real time duration, it should be tied to a physically meaningful baseline such as RTT.

Let $\mathrm{RTT}_{\mathrm{sm}}$ be a smoothed RTT estimate supplied by the transport or overlay. A robust envelope scale is then

$$U \approx c_U \cdot \mathrm{RTT}_{\mathrm{sm}},$$

where $c_U \in [0.5, 2.0]$ depends on the desired responsiveness. This keeps the response window aligned with real path dynamics and avoids interpreting D or $A_n - S_n$ as physical delays.

Clamp parameter from observed variability. The clamp parameter δ limits how far the projected release time $S_n + D + J$ may exceed the recovery time A_n. Since D is an abstract offset, δ must depend only on quantities that are time-meaningful at the receiver.

Let $\Delta_n = A_n - A_{n-1}$ denote inter-recovery spacings on the receiver clock, and let q_p be the empirical p-th percentile over a sliding window. A practical clamp choice is

$$\delta \approx c_\delta \cdot (q_{95} - q_{50}),$$

which scales allowable additional waiting to the spread between median and tail inter-recovery intervals. Updating this estimate slowly (e.g., once per several hundred objects) prevents the clamp from reacting to short-lived spikes.

Adaptive guard window from reordering statistics. The guard parameter G in QADC-G bounds waiting for predecessor objects when recovery occurs out of order. Reordering is reflected directly in recovery-time differences.

Define the skew

$$R_n = A_n - A_{n-1},$$

and treat $R_n < 0$ as reordering events. Let q_p^{reorder} be the empirical p-th percentile of $|R_n|$ for these events. A suitable guard window is

$$G \approx q_p^{\text{reorder}},$$

for example using $p = 95$, which covers most reorderings while bounding worst-case waiting.

Interaction with ADC shaping parameters. The adaptive choices for U, δ, and G are orthogonal to the shaping parameters (ρ_u, ρ_ℓ) and damping parameters $(\lambda_u, \lambda_\ell)$. The exponent parameters determine overshoot and undershoot behavior, while the adaptive parameters set the time scale on which the behavior operates.

This separation enables a practical deployment strategy:

- tune $(\rho_u, \rho_\ell, \lambda_u, \lambda_\ell)$ once per system,
- adapt (U, δ, G) automatically from RTT and inter-recovery statistics at each receiver.

Influence of application-level constraints. In addition to RTT and recovery dynamics, systems may expose application-level signals that influence desirable operating points. Examples include latency budgets, frame deadlines, control-loop sensitivity, or other application requirements that define acceptable waiting or variability. These signals may be incorporated into the adaptation logic to bias parameters toward more aggressive or more conservative smoothing.

Such influences are intentionally kept abstract here: the ADC mechanisms operate independently of any specific application semantics, and the adaptation rules above function without assuming any particular higher-level policy. However, where application constraints exist, they can guide the automatic selection of U, δ, and G to balance responsiveness and latency in a manner consistent with the needs of the deployment.

9 Algorithms

We present concise pseudocode for ADC, QADC, and QADC-G. Parameters and symbols follow the definitions above. The listings show per-object updates and scheduling and can be implemented in user space or offloaded to firmware.

ADC: Adaptive Offset

Algorithm 1 ADC: adaptive offset update and scheduling

Require: damping parameters $\lambda_u, \lambda_\ell > 0$; exponents $\rho_u, \rho_\ell \geq 0$; clipping $U > 0$; neutral band $J \geq 0$; max delay $\delta > 0$; idle timeout $T_{\text{idle}} > 0$

State: offset D; last recovery time τ_{last}

Initialize $D \leftarrow A_0 - S_0,\ \tau_{\text{last}} \leftarrow A_0$
for each recovered object with (S_n, A_n) **do**
  if $A_n - \tau_{\text{last}} \geq T_{\text{idle}}$ **then** ▷ idle re-anchor
    $D \leftarrow A_n - S_n$
  end if
  $\tau_{\text{last}} \leftarrow A_n$; $X \leftarrow A_n - (S_n + D)$
  if $X > 0$ **then**
    $Y \leftarrow \left(\frac{\min\{X,U\}}{U}\right)^{\rho_u} \cdot U$; $D \leftarrow D + \lambda_u \cdot Y$
  else if $X \leq -J$ **then**
    $Y \leftarrow -\left(\frac{\min\{|X+J|,U\}}{U}\right)^{\rho_\ell} \cdot U$; $D \leftarrow D + \lambda_\ell \cdot Y$
  end if
  $D \leftarrow \min(D,\ A_n - S_n + \delta)$
  $T_n \leftarrow \max\{A_n,\ S_n + D + J\}$ ▷ schedule
  enqueue for release at T_n
end for

QADC: Quantized Offset

Algorithm 2 QADC: quantized scheduling with midpoint hysteresis

Require: parameters of ADC plus step $\gamma > 0$

Initialize $D \leftarrow A_0 - S_0$, $E \leftarrow D + \gamma/2$
for each recovered object **do**
  update D as in ADC
  if $D > E$ **or** $D < E - \gamma$ **then**
    $E \leftarrow D + \gamma/2$
  end if
  $T_n \leftarrow \max\{A_n,\ S_n + E + J\}$; enqueue for release at T_n
end for

QADC-G: Quantized with Bounded Guard

Algorithm 3 QADC-G: ordered release with guard G

Require: parameters of QADC plus guard $G \geq 0$

for each recovered object n **do**
  compute T_n from QADC
  if $n-1$ released by T_n **then** release n at T_n
  else if $n-1$ releases in $[T_n,\, T_n + G)$ **then** release n immediately after $n-1$
  else release n at $T_n + G$
  end if
end for

10 Implementation Notes

The scheduler is transport agnostic. It consumes recovery events and sender timestamps and can run above TCP or QUIC, inside a tunnel, or alongside coded object recovery. Per-object work consists of a small number of arithmetic operations. In practice it is useful to exclude small control objects from smoothing via a minimum size threshold and release them immediately.

Representative static settings.

$$\rho_u \in [0.3, 0.7], \quad \rho_\ell \in [1.0, 2.0], \quad \lambda_u \in [0.10, 1.0], \quad \lambda_\ell \in [0.01, 0.10],$$
$$U \in [50, 200] \text{ ms}, \quad \delta \in [30, 100] \text{ ms}, \quad \gamma \in [8, 20] \text{ ms},$$
$$T_{\text{idle}} \in [0.5, 2.0] \text{ s}, \quad J \in [0, 5] \text{ ms}, \quad G \in [20, 200] \text{ ms}.$$

11 Integration of QADC into the BitRipple Tunnel

The receiver-side scheduling mechanisms in this paper are implemented inside the BitRipple Tunnel (BRT) software [1], which operates across heterogeneous paths and can employ deterministic spraying for robustness [9]. The tunnel sends objects as forward-error-protected blocks and reconstructs them at the receiver once enough coded packets have arrived. Receiver-side scheduling then determines when each recovered object is released to the application using the QADC algorithm.

Placement in the receive path. Coded packets are generated at the sender, forwarded through the tunnel, and decoded at the receiver into complete objects. The receiver-side scheduling module runs immediately after decoding and before the application interface. Each recovered object carries its sender timestamp S_n and recovery time A_n; the controller computes a release time T_n and releases the object when the receiver clock reaches T_n. This isolates timing control from lower-layer variability and bases decisions on observed recovery timing.

Operational integration and benefits. Decoding completion times reflect combined effects of path latency, congestion, and coding overhead. Immediate release produces irregular inter-arrival intervals even when throughput is stable. QADC smooths these completion events without additional coordination, maintaining a stable cadence for playback or upstream pipelines. The module is independent of the application's transport protocol and adds constant-time work per recovered object.

12 Evaluation

The evaluation comprises three entirely independent studies that use distinct data sources, methodologies, and environments. The first study reports results from a commercial operator trial comparing BRT+QADC against the native cloud-gaming stack under matched conditions. The second study analyzes an offline object-release trace captured from an Nvidia GeForce NOW session and applies the receiver-side scheduling algorithm to the same underlying traffic to compare smoothed and unsmoothed release behavior. The third study evaluates ADC on a synthetic recovery pattern designed to reflect short-scale jitter characteristics observed in practice, which isolates the algorithmic behavior under controlled delay variation. Each study stands on its own, with separate data sources and experimental setups, and together they highlight different aspects of receiver-side scheduling behavior.

12.1 Telecommunications Operator Cloud-Gaming Trial

An evaluation was conducted in collaboration with a major telecommunications operator using a cloud-gaming workload. The operator executed controlled A/B tests comparing the BitRipple Tunnel with integrated receiver-side scheduling (BRT+QADC) against the native cloud-gaming system. The experimental setup was identical across conditions: the same cloud-gaming server, client device, access network, game title, video-encoding quality, frame rate, and scripted interaction pattern were used throughout the A/B comparison. When BRT+QADC was enabled, a tunnel was created between the client and an edge-site server, and all native traffic flowed through this tunnel unmodified at the application layer. When BRT+QADC was disabled, traffic flowed directly between the client and the cloud-gaming server. This ensured that any performance differences were attributable solely to the presence or absence of the BRT+QADC path and its integrated scheduling module.

Each set of experiments comprised multiple two-minute sessions that alternated between configurations so that both conditions experienced comparable network conditions. Tests were repeated at multiple times of day to capture naturally occurring variation in cellular and Wi-Fi channel quality. All timing and frame-delivery statistics were collected by an external analytics environment instrumenting both the client and the cloud servers.

Representative Video Evidence. Two example sessions illustrate the perceptual difference introduced by BRT+QADC. These were back-to-back two-minute runs of a 40 Mbps, 120 fps cloud-gaming workload executed under near-identical network conditions. The first video shows the session with BRT+QADC enabled [3], and the second shows the native configuration [4]. The BRT+QADC-enabled session exhibits visibly smoother and more consistent playback. These particular videos come from a different workload than the one summarized in Table 1, but were collected under the same overall A/B methodology.

Representative Percentiles. Table 1 reports 95th- and 99th-percentile statistics for a representative set of eight two-minute sessions collected using a 28 Mbps, 60 fps workload. As in the larger trial design, the sessions were executed back-to-back and alternated between the two configurations, yielding four BRT+QADC sessions and four Native sessions under comparable mixed cellular and Wi-Fi conditions. Each row corresponds to one session, and the rows can be interpreted as repeated trials conducted in the same controlled experimental environment.

The BRT+QADC configuration maintains tight inter-frame arrival times (IAT) and avoids the extreme round-trip time (RTT) outliers observed in the Native configuration. Tail latency decreases

by more than a factor of three, reflecting the combined effect of BRT and the integrated scheduling on long delay excursions while preserving low average latency.

At 60 fps, each frame interval is approximately 16.7 ms, so percentile timing directly reflects perceptual smoothness. A deviation beyond the 95th percentile affects at least 5% of intervals (roughly three frames per second on average), and deviations beyond the 99th percentile occur in at least 1% of intervals. Such events often occur in bursts rather than being evenly spaced, so large excursions can create noticeable jitter clusters during gameplay.

Table 1: Representative percentiles for a 28 Mbps, 60 fps workload over mixed cellular and Wi-Fi paths. Comparison between the BRT+QADC configuration and the Native configuration.

Configuration	IAT [ms]		RTT [ms]	
	95th	99th	95th	99th
BRT+QADC	20.1	20.1	112.0	132.5
BRT+QADC	19.6	19.6	105.0	120.0
BRT+QADC	19.5	19.5	107.0	119.0
BRT+QADC	19.4	19.7	101.5	112.0
Native	29.9	70.3	47.0	362.0
Native	33.9	86.6	87.0	424.6
Native	29.5	70.5	61.0	112.5
Native	39.3	92.8	116.6	469.8

Across the trials, BRT+QADC consistently reduced large jitter excursions and improved release cadence without measurable throughput penalty. Because the controller operates solely on receiver-observed recovery timing, it compensates for variable decoding and path latency on a per-object basis while leaving congestion control and coding efficiency unchanged. The results demonstrate that receiver-side scheduling provides an effective software-only mechanism for achieving tighter timing stability in field conditions.

12.2 Offline Trace Evaluation: GeForce NOW Release Schedule

We evaluated receiver-side scheduling on an object-release trace captured from an NVIDIA GeForce NOW session. The trace contains the platform's native release timing for each object; we applied ADC/QADC offline to the same sequence to generate a smoothed release schedule.

Figure 5 shows the deviations between the release times and a simple receiver-side playback buffer model applied to this trace. Positive excursions represent instantaneous lags in the buffer's output timing, and negative excursions represent jumps.

The unsmoothed stream (dark gray trace) exhibits frequent and large excursions: approximately 18,915 positive and 18,995 negative events over the displayed sequence. When the same input trace is processed by the receiver-side scheduling algorithm (light gray trace), these excursions are reduced to 2,590 positive and 2,669 negative events. The smoothed timeline therefore minimizes oscillations in the buffer and yields a far more stable release pattern, consistent with perceptually smooth playback under fluctuating network conditions.

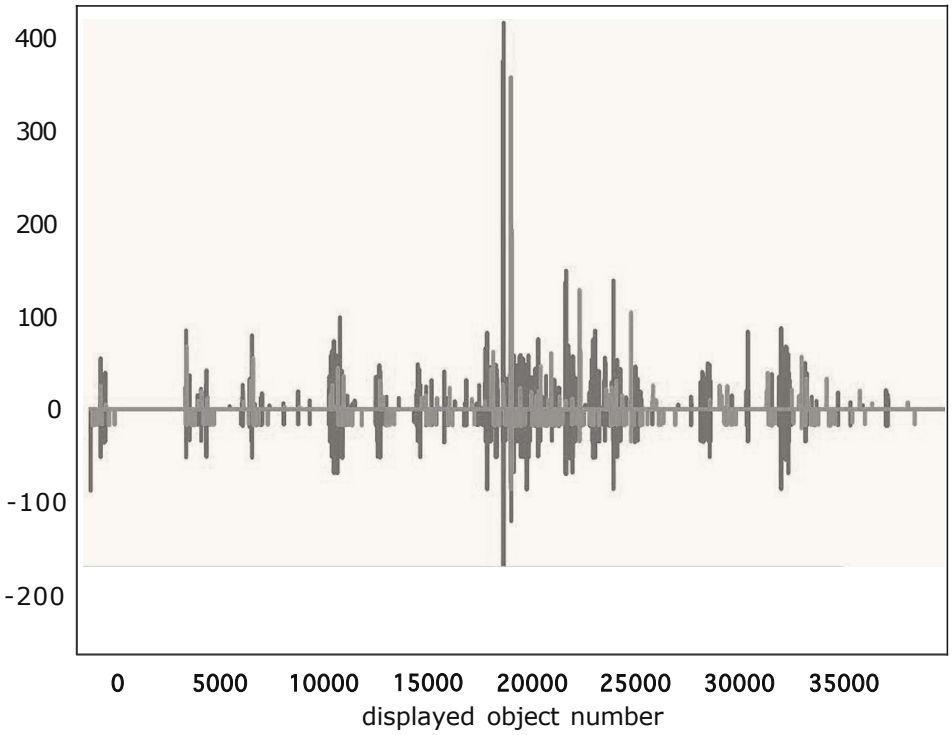

Figure 5: Playback buffer deviation events for a cloud-gaming trace with and without smoothing.

12.3 Synthetic Recovery-Pattern Evaluation

To isolate the effects of receiver-side scheduling under controlled timing variability, we evaluated ADC/QADC on a synthetically generated recovery sequence. The synthetic trace was constructed to exhibit representative short-scale jitter patterns observed in practice, alternating modest delay valleys and elevated delay peaks, while preserving a constant sending cadence for consecutive objects at the sender.

Figure 6 presents a percentile comparison between the smoothed and unsmoothed timing derived from this synthetic trace. The solid curve shows the additional release delay introduced by the scheduler relative to each object's recovery time. For most objects this added delay remains small, increasing only at the upper percentiles where the scheduler compensates for large positive deviations in the synthetic recovery pattern.

The dashed and dotted curves compare the inter-send time between consecutive objects at the sender to the corresponding inter-release time at the receiver, showing how closely the smoothed schedule reproduces the original sending cadence. The dotted curve corresponds to the unsmoothed (original) recovery sequence, which exhibits wide percentile spread and significant timing distortion. The dashed curve corresponds to the smoothed sequence, which remains tightly clustered near zero across nearly the entire percentile range. This demonstrates that the receiver-side scheduler restores a nearly uniform cadence while applying only bounded delays.

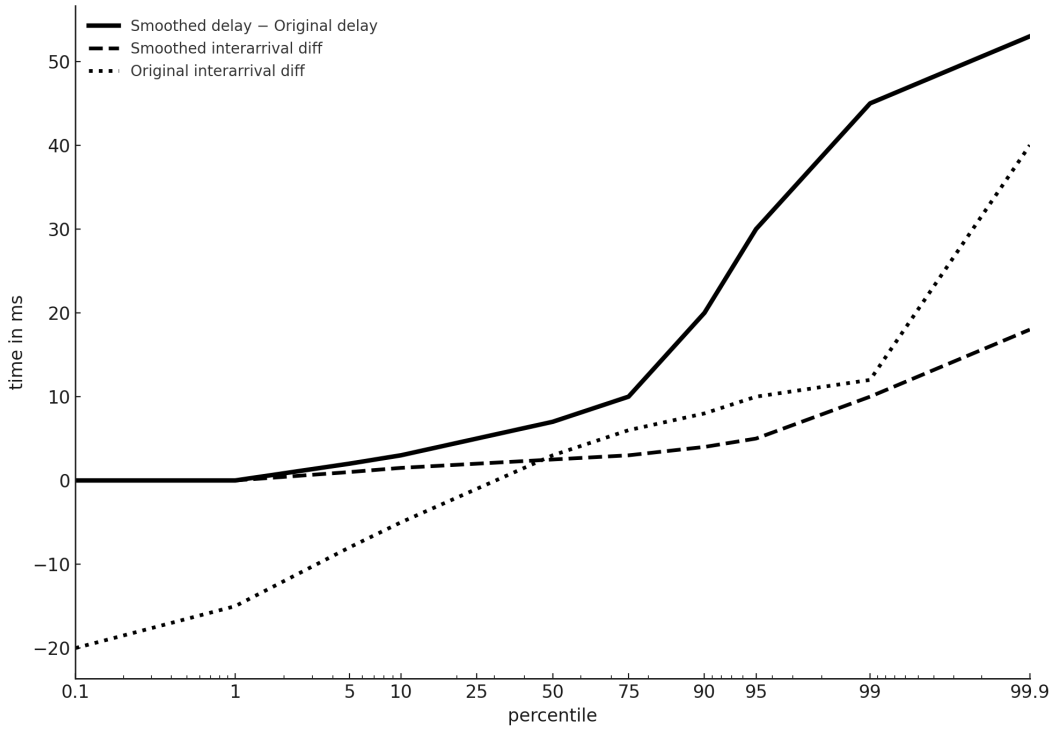

Figure 6: Representative percentile comparison between smoothed and unsmoothed release timing.

13 Analytic Analysis and Concrete Examples

To understand the behavior of the Adaptive Delay Control (ADC) scheduler, it is valuable to analyze its evolution on simple alternating delay patterns. In this section all quantities are normalized by setting the current peak delay to

$$d_{\mathrm{H}} = 1, \qquad d_{\mathrm{L}} = 1 - g, \qquad 0 < g < 1,$$

and choosing the shaping scale to match the peak,

$$U = 1.$$

This corresponds to autoscaling the update rule so that deviations are measured relative to the current round trip time or current one way delay. All delays are therefore expressed in normalized units.

For each object n the sender time is $S_n = n \cdot \tau$ for a fixed time interval $\tau > 0$. The one way delay alternates between the normalized peak and valley,

$$d_n = \begin{cases} 1, & n \text{ even}, \\ 1 - g, & n \text{ odd}, \end{cases}$$

so the recovery time is $A_n = S_n + d_n$. ADC maintains an adaptive offset D_n and forms the projected release time $R_n = S_n + D_n$. The release time is

$$T_n = \max\{A_n,\ R_n\}.$$

The deviation that drives the update is

$$X_n = d_n - D_n.$$

Throughout this section the neutral band is $J = 0$ and the clamp δ is large enough to be inactive. The focus is on the fixed point behavior of the adaptive offset when the step size λ is small.

13.1 Asymptotic Analysis as $\lambda \to 0$

When λ is small the offset moves only slightly on each iteration. In steady state the value of D_n converges to a value D as $\lambda \to 0$. Write the value of D as:

$$D = 1 - \Delta, \qquad 0 < \Delta < g,$$

so that Δ is the distance below the peak delay.

Linear asymptote (length-2 pattern): For the linear model

$$\rho_u = \rho_\ell = 1, \qquad Y_n = X_n,$$

and the update rule is

$$D_{n+1} = \begin{cases} D_n + \lambda_u \cdot (1 - D_n), & n \text{ even}, \\ D_n + \lambda_\ell \cdot ((1 - g) - D_n), & n \text{ odd}. \end{cases}$$

With asymmetric gains

$$\lambda_u = (1 - \varepsilon) \cdot \lambda, \qquad \lambda_\ell = \varepsilon \cdot \lambda, \qquad 0 < \varepsilon < 1,$$

the net change across one high and one low step must be zero in steady state. Setting $D = 1 - \Delta$ and equating the upward and downward movements yields

$$(1 - \varepsilon) \cdot \Delta = \varepsilon \cdot (g - \Delta). \tag{1}$$

Solving for Δ gives

$$\Delta = \varepsilon \cdot g, \qquad D = 1 - \varepsilon \cdot g.$$

This confirms that the linear update stabilizes at a point between the peak and the valley, displaced $\varepsilon \cdot g$ below the peak and $(1 - \varepsilon) \cdot g$ above the valley. Unless ε is extremely small, the offset remains noticeably below the upper envelope and can never approach 1 closely in the $\lambda \to 0$ regime.

Asymmetric exponent asymptote (length-2 pattern): The full ADC update applies exponent shaping,

$$Y_n = \begin{cases} X_n^{\rho_u}, & X_n > 0, \\ -|X_n|^{\rho_\ell}, & X_n < 0, \end{cases} \qquad U = 1,$$

so on high delay steps the deviation is $X_n = 1 - D_n$ and the change in D_n is proportional to $(1 - D_n)^{\rho_u}$. On low delay steps the deviation is $X_n = (1 - g) - D_n$ and the change is proportional to $-(D_n - (1 - g))^{\rho_\ell}$.

Write the steady state offset as

$$D = 1 - \Delta, \qquad 0 < \Delta < g,$$

so that the distance below the peak is Δ and the distance above the valley is $g - \Delta$. On a high delay step the update magnitude is Δ^{ρ_u}, and on a low delay step it is $(g - \Delta)^{\rho_\ell}$.

With asymmetric gains

$$\lambda_u = (1 - \varepsilon) \cdot \lambda, \qquad \lambda_\ell = \varepsilon \cdot \lambda, \qquad 0 < \varepsilon < 1,$$

the net change in D across one high and one low step in the $\lambda \to 0$ regime must be zero at the fixed point. The balance condition is

$$(1 - \varepsilon) \cdot \lambda \cdot \Delta^{\rho_u} = \varepsilon \cdot \lambda \cdot (g - \Delta)^{\rho_\ell},$$

which is equivalent to Equation (1) in the special case when $\rho_u = \rho_\ell = 1$. Canceling $\lambda > 0$ and rearranging gives the fixed point equation

$$\Delta^{\rho_u} = \frac{\varepsilon}{1 - \varepsilon} \cdot (g - \Delta)^{\rho_\ell}.$$

This equation has a unique solution in $(0, g)$. It also yields a useful upper bound on Δ. Since $0 < g - \Delta < g$, we have

$$(g - \Delta)^{\rho_\ell} < g^{\rho_\ell},$$

and therefore

$$\Delta^{\rho_u} = \frac{\varepsilon}{1 - \varepsilon} \cdot (g - \Delta)^{\rho_\ell} < \frac{\varepsilon}{1 - \varepsilon} \cdot g^{\rho_\ell}.$$

Taking the ρ_uth root gives

$$0 < \Delta < \left(\frac{\varepsilon}{1 - \varepsilon}\right)^{1/\rho_u} \cdot g^{\rho_\ell/\rho_u}.$$

Thus the steady state offset satisfies

$$D = 1 - \Delta > 1 - \left(\frac{\varepsilon}{1 - \varepsilon}\right)^{1/\rho_u} \cdot g^{\rho_\ell/\rho_u}.$$

For the exponent asymmetry of interest, with $0 < \rho_u < 1 < \rho_\ell$ and $g < 1$, the exponent ρ_ℓ/ρ_u is greater than one, which makes g^{ρ_ℓ/ρ_u} very small. In typical operating regimes $\varepsilon < 1/2$, so $\varepsilon/(1 - \varepsilon)$ is less than one. Since $1/\rho_u > 1$, raising this ratio to the power $1/\rho_u$ reduces it even further. Consequently,

$$\left(\frac{\varepsilon}{1 - \varepsilon}\right)^{1/\rho_u} \cdot g^{\rho_\ell/\rho_u}$$

is extremely small, and the equilibrium offset Δ lies very close to the peak value $d_{\mathrm{H}} = 1$.

In the special case $\varepsilon = \frac{1}{2}$ this bound reduces to

$$\Delta < g^{\rho_\ell/\rho_u},$$

which matches the symmetric step size result.

13.2 Why λ and ε Cannot Be Too Small

Although the $\lambda \to 0$ regime yields clean analytical expressions and sharp upper envelope tracking under exponent asymmetry, choosing λ too small is undesirable in practice. The rate at which the offset approaches its steady state is governed by a contraction whose speed is proportional to λ, so the convergence time grows on the order of $1/\lambda$. In realistic network settings the path delay can fluctuate or the sending rate can shift on short timescales, and very small values of λ cause the scheduler to react too slowly to such global changes.

Similarly, ε cannot be chosen too small. When objects are consistently recovered early after an upward fluctuation in the projected release time, the downward correction is driven by the low step with gain $\lambda_\ell = \varepsilon \cdot \lambda$. If ε is too small, the projected release time decreases only very slowly, causing extended periods during which early recoveries do not meaningfully reduce the offset. This slows the reaction to downward trends in recovery times and reduces the scheduler's ability to realign with the upper envelope.

In general there is a tradeoff between λ and ε. Smaller λ slows global responsiveness, while smaller ε slows downward adjustments following early recoveries. Both parameters should therefore be kept large enough to ensure timely reactions to changing conditions, while still small enough to smooth the inter-release times of objects and maintain a stable release cadence.

13.3 Concrete Examples

To illustrate the behavior in realistic units, we interpret $d_{\mathrm{H}} = 1.0$ as a delay of 100 ms and $d_{\mathrm{L}} = 0.7$ as 70 ms, so the gap is

$$g = d_{\mathrm{H}} - d_{\mathrm{L}} = 30\,\mathrm{ms}.$$

We also take the inter-send interval to be

$$\tau = 16.7\,\mathrm{ms},$$

corresponding to a 60 fps video stream in which each object is a frame. In all concrete examples below we fix

$$\lambda = 0.2, \qquad \varepsilon = 0.2,$$

initialize the offset according to the first recovered object,

$$D_0 = d_{\mathrm{H}} = 1.0 \longrightarrow 100\,\mathrm{ms},$$

and then iterate the update rules for a long sequence of objects until the behavior becomes periodic. For the exponent-asymmetry cases we use the same pair of exponents throughout:

$$\rho_u = 0.5, \qquad \rho_\ell = 2.0.$$

Linear example (length-2 pattern): For the alternating pattern $S = 2$ the nominal delays are

$$d_n = \begin{cases} 100\,\mathrm{ms}, & n \text{ even}, \\ 70\,\mathrm{ms}, & n \text{ odd}. \end{cases}$$

The gains are

$$\lambda_u = (1 - \varepsilon) \cdot \lambda = 0.16, \qquad \lambda_\ell = \varepsilon \cdot \lambda = 0.04.$$

Running the linear update with the sign-aware rule

$$D_{n+1} = \begin{cases} D_n + \lambda_u \cdot (100\,\text{ms} - D_n), & X_n > 0, \\ D_n + \lambda_\ell \cdot (70\,\text{ms} - D_n), & X_n < 0, \end{cases} \qquad X_n = d_n - D_n,$$

produces a stable two-point cycle for the effective offset at high and low steps. In steady state the values are approximately

$$D_{\text{H}} \approx 93.8\,\text{ms}, \qquad D_{\text{L}} \approx 94.8\,\text{ms}.$$

On high-delay steps the release follows the arrival,

$$T_n - S_n = 100\,\text{ms},$$

while on low-delay steps the release is governed by the offset,

$$T_n - S_n \approx 94.8\,\text{ms}.$$

The inter-release intervals therefore alternate between

$$T_n - T_{n-1} \approx 21.9\,\text{ms} \quad \text{and} \quad T_n - T_{n-1} \approx 11.5\,\text{ms}.$$

The linear scheduler produces a strongly uneven cadence at the receiver, with every other interval substantially shorter than τ and the others substantially longer.

Asymmetric exponent example (length-2 pattern): With exponent shaping

$$\rho_u = 0.5, \qquad \rho_\ell = 2.0,$$

and the same λ and ε, the update becomes

$$D_{n+1} = \begin{cases} D_n + \lambda_u \cdot (X_n)^{\rho_u}, & X_n > 0, \\ D_n - \lambda_\ell \cdot (-X_n)^{\rho_\ell}, & X_n < 0, \end{cases} \qquad X_n = d_n - D_n.$$

Starting from $D_0 = 100\,\text{ms}$ and iterating, the system converges to a short-period cycle in which the effective delays

$$T_n - S_n = \max\{d_n,\ D_n\}$$

remain very close to the peak. In steady state the releases for both high and low steps lie in a narrow band between approximately

$$97.9\,\text{ms} \quad \text{and} \quad 100.5\,\text{ms},$$

with low-delay arrivals occasionally being held slightly past the nominal peak due to overshoot in the offset.

The resulting inter-release intervals follow a four-element pattern in steady state:

$$T_n - T_{n-1} \approx \{16.3,\ 16.7,\ 16.5,\ 17.2\}\,\text{ms},$$

repeating. All intervals stay close to the nominal $\tau = 16.7\,\text{ms}$, and the jitter is limited to about $\pm 0.5\,\text{ms}$. The exponent asymmetry therefore yields an almost uniform release cadence, in contrast to the pronounced two-point oscillation under the linear update.

13.4 Extension to longer recovery patterns

The alternating pattern analyzed above corresponds to pattern length 2, in which each late recovery is immediately followed by an early recovery. In realistic environments, however, late recoveries often appear singly, while early recoveries occur in consecutive sequences. This motivates extending the analysis to a length-P pattern in which one object has peak delay $d_{\mathrm{H}} = 1$ and the remaining $P-1$ objects have valley delay $d_{\mathrm{L}} = 1-g$. The pattern then repeats every P steps.

As before, write the steady-state offset in the small-λ regime as

$$D = 1 - \Delta, \qquad 0 < \Delta < g.$$

Linear asymptote (length-P pattern): With asymmetric gains

$$\lambda_u = (1-\varepsilon)\cdot\lambda, \qquad \lambda_\ell = \varepsilon\cdot\lambda,$$

and symmetric exponents $\rho_u = \rho_\ell = 1$, the fixed point satisfies

$$(1-\varepsilon)\cdot\Delta = (P-1)\cdot\varepsilon\cdot(g-\Delta).$$

Solving for Δ gives the linear steady-state distance below the peak:

$$\Delta = \frac{(P-1)\cdot\varepsilon}{1+(P-2)\cdot\varepsilon}\cdot g, \qquad D = 1-\Delta.$$

This reduces to $\Delta = \varepsilon\cdot g$ when $P = 2$. As P increases, the influence of the valley steps accumulates and the fixed point moves further into the valley.

Asymmetric exponent asymptote (length-P pattern): With exponent shaping,

$$Y_n = \begin{cases} X_n^{\rho_u}, & X_n > 0, \\ -|X_n|^{\rho_\ell}, & X_n < 0, \end{cases} \qquad 0 < \rho_u < 1 < \rho_\ell,$$

the balance over one period becomes

$$(1-\varepsilon)\cdot\Delta^{\rho_u} = (P-1)\cdot\varepsilon\cdot(g-\Delta)^{\rho_\ell}.$$

This equation has a unique solution in $(0, g)$. A convenient upper bound follows from $0 < g-\Delta < g$:

$$\Delta < \left(\frac{(P-1)\cdot\varepsilon}{1-\varepsilon}\right)^{1/\rho_u}\cdot g^{\rho_\ell/\rho_u}, \qquad D = 1-\Delta.$$

This bound is informative only when the multiplicative factor $(P-1)\cdot\varepsilon/(1-\varepsilon)$ is less than one. Solving

$$\frac{(P-1)\cdot\varepsilon}{1-\varepsilon} < 1$$

gives the condition $\varepsilon < 1/P$. When ε satisfies this inequality, the right-hand side of the bound is strictly smaller than g, and the quantity

$$\left(\frac{(P-1)\cdot\varepsilon}{1-\varepsilon}\right)^{1/\rho_u}\cdot g^{\rho_\ell/\rho_u}$$

becomes very small in the regime $0 < \rho_u < 1 < \rho_\ell$, implying that Δ lies close to the peak. When $\varepsilon \geq 1/P$, the bound no longer provides a meaningful guarantee.

These expressions generalize the $P = 2$ results and show that exponent asymmetry continues to keep the offset near the peak even when many consecutive valley recoveries occur, provided ε is not too large and g^{ρ_ℓ/ρ_u} remains small.

13.5 Concrete example for length-10 pattern

To illustrate the effect of an extended recovery pattern, consider the case $P = 10$, corresponding to one late recovery followed by nine early recoveries. We again interpret

$$d_{\mathrm{H}} = 1.0 \longrightarrow 100\,\mathrm{ms}, \qquad d_{\mathrm{L}} = 0.7 \longrightarrow 70\,\mathrm{ms}, \qquad g = 30\,\mathrm{ms},$$

and use the same inter-send interval

$$\tau = 16.7\,\mathrm{ms},$$

the same gains $\lambda = 0.2$, $\varepsilon = 0.2$, and the same initialization $D_0 = 100\,\mathrm{ms}$. One object in each 10-object period has delay 100 ms; the remaining nine have delay 70 ms.

Linear example (length-10 pattern): As before, the gains on upward and downward corrections are

$$\lambda_u = (1 - \varepsilon) \cdot \lambda = 0.16, \qquad \lambda_\ell = \varepsilon \cdot \lambda = 0.04.$$

On each step the update

$$D_{n+1} = \begin{cases} D_n + \lambda_u \cdot (100\,\mathrm{ms} - D_n), & X_n > 0, \\ D_n + \lambda_\ell \cdot (70\,\mathrm{ms} - D_n), & X_n < 0, \end{cases} \qquad X_n = d_n - D_n,$$

is applied using the sign of the deviation X_n, regardless of whether the nominal step is a peak or a valley.

After a long warm-up, the effective delays $T_n - S_n = \max\{d_n, D_n\}$ stabilize into a 10-point cycle. In one representative period the values are

$$T_n - S_n \approx \{100.0,\ 81.5,\ 81.0,\ 80.6,\ 80.3,\ 79.9,\ 79.6,\ 79.2,\ 78.8,\ 78.3\}\,\mathrm{ms},$$

where the first element corresponds to the late step and the remaining nine to the early steps. The offset continues to drift downward over the early steps, so valley objects are released progressively closer to the raw 70 ms arrival.

The inter-release intervals over the same 10-object period are

$$T_n - T_{n-1} \approx \{38.4,\ -1.8,\ 16.2,\ 16.3,\ 16.3,\ 16.3,\ 16.3,\ 16.3,\ 16.3,\ 16.4\}\,\mathrm{ms}.$$

The negative interval indicates that one object is scheduled to be released slightly *before* its predecessor, so the raw update rule can generate out-of-order releases. In steady state the late object induces a long inter-release gap of almost 40 ms, more than 20 ms above the nominal inter-send interval $\tau = 16.7\,\mathrm{ms}$; this large lag then forces the next object to be scheduled roughly 2 ms before its predecessor, with the remaining eight intervals clustered near 16.3 ms.

Exponent asymmetry example (length-10 pattern): With exponent shaping

$$\rho_u = 0.5, \qquad \rho_\ell = 2.0,$$

and the same λ and ε, the update is

$$D_{n+1} = \begin{cases} D_n + \lambda_u \cdot (X_n)^{\rho_u}, & X_n > 0, \\ D_n - \lambda_\ell \cdot (-X_n)^{\rho_\ell}, & X_n < 0, \end{cases} \qquad X_n = d_n - D_n.$$

In steady state the effective delays again form a 10-point cycle. A representative period has

$$T_n - S_n \approx \{100.0,\ 99.6,\ 99.3,\ 98.9,\ 98.6,\ 98.3,\ 98.0,\ 97.6,\ 97.3,\ 97.0\}\,\text{ms},$$

with the first value associated with the late step and the others with the nine early steps. Despite each early recovery arriving 30 ms before the peak, the projected release times for valley objects stay within a few milliseconds of the peak.

The corresponding inter-release intervals over the same period are

$$T_n - T_{n-1} \approx \{19.7,\ 16.3,\ 16.3,\ 16.4,\ 16.4,\ 16.4,\ 16.4,\ 16.4,\ 16.4,\ 16.4\}\,\text{ms}.$$

Aside from a single interval at the start of each period that is extended to about 19.7 ms (a modest additional delay of roughly 3 ms beyond τ), all intervals lie very close to $\tau = 16.7$ ms, with variations of at most a few tenths of a millisecond. There are no negative intervals, so the release order is preserved without any additional guard.

Even with nine consecutive early recoveries per cycle, exponent asymmetry maintains tight upper-envelope tracking and a nearly uniform release cadence. In contrast, the linear update rule produces large oscillations and can schedule objects out of order, with the severity of the pattern increasing as P grows.

14 Discussion and Limitations

The asymmetric response can retain residual latency after a delay spike, since it decreases more slowly during the subsequent valley; the clamp parameter δ limits how much of this residual drift can accumulate. The quantization step γ should be commensurate with the application cadence to avoid visible stepping. The guard G improves safety for in-order transports at the cost of small delays when predecessors are late.

Receiver-side scheduling is complementary to transport-level mechanisms. It does not replace congestion control, retransmission, or existing jitter buffers, but instead adds a timing layer that operates on recovered objects. Exploring joint design with transport algorithms and automated parameter selection based on observed trace statistics are natural extensions of this work.

15 Conclusion

Receiver-side scheduling stabilizes object release cadence with minimal complexity and without sender coordination. The adaptive offset, quantized hysteresis, and bounded in-order guard together provide low-jitter, low-latency behavior across variable network conditions. Experiments with a cloud-gaming workload show that integrating receiver-side scheduling into an application-layer overlay can remove virtually all large jitter excursions and significantly tighten inter-release timing. These results suggest broader applicability to interactive streaming and related workloads that require tight timing at the receiver.

References

[1] Pooja Aggarwal, Michael Luby, and Lorenz Minder. Enabling immersive experiences in challenging network conditions. *arXiv preprint arXiv:2304.03732v2*, 2025.

[2] S. Bhunia and S. Mukherjee. An adaptive jitter buffer playout algorithm for enhanced voip performance. Technical report, ACity Conference, 2011.

[3] BitRipple. Cloud Gaming Trial with BitRipple Tunnel. YouTube video, 2025. Available at https://www.youtube.com/watch?v=o0dk_5OttHk.

[4] BitRipple. Cloud Gaming Trial without BitRipple Tunnel. YouTube video, 2025. Available at https://www.youtube.com/watch?v=Jgp7zLmzRuk.

[5] Haivision. *Haivision Media Gateway Protocol Settings*, 2023.

[6] Haivision. *Latency (SRT 1.5.3 Documentation)*, 2023.

[7] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of ACM SIGCOMM*, 2014.

[8] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, and H. Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *Proceedings of ACM CoNEXT*, 2012.

[9] Michael Luby and John Byers. Whack-a-mole: Deterministic packet spraying across multiple network paths. *arXiv preprint arXiv:2509.18519v1*, 2025.

[10] S. Maity, S. Chakraborty, and R. Sarkar. Qos enhancement using an adaptive jitter buffer algorithm. In *Advances in Communication, Devices and Networking*. Springer, 2018.

[11] A. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinne. Adaptive playout mechanisms for packetized audio applications in wide-area networks. In *Proceedings of IEEE INFOCOM*, 1994.

[12] C. Rose, M. Yajnik, J. Kurose, and D. Towsley. Integrating packet fec into adaptive voice playout buffer algorithms on the internet. Technical report, University of Massachusetts Amherst, 2000.

[13] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. Rtp: A transport protocol for real-time applications. RFC 3550, 2003.

[14] M. Sharabayko et al. The srt protocol. IETF Internet-Draft draft-sharabayko-srt-02, 2021.

[15] P. Singer and S. Desineni. Transmission time offsets in rtp streams. RFC 5450, 2009.