Real-time Game–Broadcast Guidance Protocol for Spoiler Prevention in Live Game Streaming

YongJoon Joe*, Sung-Il Jang and DongMyung Shin

LSware Inc., Seoul, South Korea
E-mail: eungene@lsware.com; sungil@lsware.com; roland@lsware.com
**Corresponding Author*

Received 01 September 2025; Accepted 01 October 2025

Abstract

Spoiling story content during live streams of narrative games can undermine creator intent and expose creators to legal and policy risks. We present a real-time game–broadcast guidance protocol driven by a license-script interpreter. A game control module and a broadcasting SW plugin exchange authenticated states; the interpreter evaluates a DSL over synchronized game and broadcast schemas to issue actions, such as in-game choice limits and stream blackouts. Unlike DRM, this approach aims to prevent harm to both rights holders and streamers arising from accidental violations of license terms. This protocol does not burden streamers by operationalizing publisher guidelines that restrict spoilers and the disclosure of endings, and it supports well-intentioned creators in avoiding unintentional breaches caused by oversight or error. On a two-machine testbed at 1080p60, 1-hour streams achieved an average end-to-end block latency of about 65 ms, with minimal CPU overhead.

Keywords: Game streaming, spoiler prevention, real-time control, copyright licensing, copyright compliance.

Journal of Web Engineering, Vol. 24_8, 1263–1282.
doi: 10.13052/jwe1540-9589.2484

1 Introduction

Live streaming boosts discovery and sales. This influence is powerful in narrative games, where word of mouth and creator reach are vital [5]. However, the same streams can reduce purchase intent when core story beats are shown [2]. This trade-off encourages precise, real-time control at spoiler boundaries instead of outright bans.

Game studios tried policy first. Atlus initially tightened and later relaxed the rules for Persona 5 Royal, ultimately allowing full-game streaming while requesting that creators label spoilers and avoid pre-release play. These changes demonstrate the marketing value of streaming and the need to balance story leaks with ongoing coverage [1, 10]. Platforms enforce copyright, not spoiler etiquette. Twitch removes content upon receiving a DMCA notice, but does not filter narrative spoilers independently [3]. Courts are beginning to view spoiler uploads as a form of market harm. In 2023, a Japanese court convicted a YouTuber for posting visual-novel endings and spoiler compilations [4].

However, human compliance is limited to the scale of the guidelines. Title-specific rules become complex with branching plots, late-game twists, and artistic misdirection. Streamers cannot keep track of dozens of conditional clauses while live streaming. This cognitive burden, which induces streamers' unintended violence, forces publishers to simplify guidance, reducing protection and still risking accidental spoilers. If a publisher needs detailed, scene-level policies, enforcement must be automatic rather than manual.

This paper introduces automation through a license-script DSL that encodes publisher policies as executable and continuously monitors synchronized game and broadcast states. When a rule detects a spoiler scene by the script interpretation, the system responds with coordinated actions: controlling in-game behavior to prevent spoilers and managing broadcast output by blacking out or muting only the spoiling span. Representative policies include date-based rollout, which releases early chapters while keeping other content blocked by default; per-scene option limits during live events; and monetization rules for streams, including late-game restrictions defined by IP holders.

The goal is to achieve precise control without a DRM-style run-stop. Existing solutions, such as Sony's spoiler-blocking patents, focus on filtering for game players, not for the audience of game streaming, based on progress metadata. Our approach complements this by gating the live source through

in-engine context, ensuring that allowed content stays on air and only spoilers are hidden. Significantly, automation eases the streamer's load: streamers no longer need to memorize and recall changing rules during play or streaming; the system guides them, reducing errors and stress while maintaining proper coverage.

In this paper, we will introduce:

- A license-script DSL and interpreter for narrative-aware, real-time guidance
- A bidirectional protocol and state model spanning game and broadcasting software
- An evaluation showing a mean of 65 ms and P95 $\leq$ 500 ms block latency with negligible overhead and zero false positives under the stated setup.

2 Background and Related Work

2.1 Platform Rules and Legal Context

Twitch enforces DMCA procedures and removes content upon receiving a notice from the rights holder. It does not classify "spoilers" without a copyright claim and cannot filter spoilers during live streaming [3]. YouTube depends on Content ID and DMCA; implementations differ by publisher and are geared toward post-processed videos rather than live filtering. In Japan, some platforms and publishers coordinate around music licensing (e.g., JASRAC) and restrict music streaming to services authorized by JASRAC. Square Enix guidelines explicitly mandate the use of JASRAC-licensed services. Nintendo's global creator guidelines allow sharing transformative gameplay after a game's release, but reserve the right to object to inappropriate content [9].

On the publisher side, Japanese companies often issue title-specific policies. Atlus's Persona 5 Royal policy evolved from strict late-game limits to broadly permitted streaming with spoiler labeling [1, 10]. These policies aim to protect story value while maximizing promotional benefits.

In 2023, Japan's Sendai District Court handed down a suspended prison sentence and a fine for posting spoiler-heavy gameplay videos of a visual novel, confirming that unauthorized public sharing of cinematic story content can be a criminal offense [4]. Along with platform DMCA practices, this encourages the use of preventative, real-time technical controls in live environments.

As shown so far, IP holders (publishers) dedicate considerable effort to developing guidelines that control what is shown during game streaming to boost sales. Also, due to these incidents, the significance of licensing has been strongly highlighted for streamers. Yet, for the "good" streamers who genuinely try to follow the rules, it has, ironically, become a heavy burden, making game streaming more difficult, decreasing the number of streams, and ultimately putting game companies at a disadvantage as well.

2.2 Console- and Service-level Spoiler Controls

Sony's Spoiler Block patents describe hiding undiscovered plot elements using developer metadata and user progress, so players avoid seeing content they have not reached. This approach focuses on enhancing the player experience, rather than live streaming or copyright enforcement [6, 7]. Also, PlayStation supports blocking capture during flagged scenes, pausing recording when a cutscene or licensed segment appears. This coarse mechanism demonstrates an in-game level blackout of sensitive content [11, 14].

Neither console capture flags nor user-side filtering coordinate the game engine and the live streaming encoder with the publisher's policy in real-time. That gap is the focus of a license-script protocol.

2.3 AI-approach to Filtering Spoiling Content

There are also some approaches [12, 13] that use an AI system to filter spoiling content. Several AI-driven approaches are being developed based on audience comments or text content. However, these approaches do not block the streaming of spoiling content; the spoiling content is already distributed and blocked on the audience side. Therefore, these approaches cannot fully assist IP holders and streamers, especially with live streaming.

2.4 Industry Guidance Patterns Relevant to DSL Design

Traditional patterns include time-based releases of permitted chapters, restrictions on late-game and ending content, allowances or bans based on platform and monetization, and caveats regarding copyright and music. Square Enix's requirement to stream only on JASRAC-licensed platforms aligns with a broadcast-style model that captures monetization, platform, and channel identity; Nintendo's focus on transformative use involves allowing gameplay while avoiding full cutscene reposts [8, 9]. The proposed rules implement these patterns as automated, executable policies: date-based

unlocking of chapters, monetization-dependent late-game blackouts, and per-scene restrictions during live broadcasts.

2.5 Positioning of This Work

Previous methods either block everything at specific moments (using engine or console flags) or filter content after playback (pre-streaming filters). The protocol here runs before frames leave the streamer's computer, relying on a deterministic game and broadcast state with an auditable set of rules. It prevents leakage at the source and eliminates the need for ongoing human rule monitoring during live play.

3 Problem Statement and Requirements

The goal is to prevent story-critical content from reaching viewers during live streams of narrative games without disrupting gameplay or the entire broadcast. Enforcement must be automatic. The protocol assesses the synchronized game and broadcast state and issues coordinated commands in real-time, so only spoiler spans are hidden while allowed content remains visible. Table 1 outlines the requirements of the problem discussed in this paper.

Availability refers to the prevention of control leaks. When a spoiler scene video starts but cannot be blocked within 5 s (not partial miss) due to misinterpretation of the license script and game/broadcast information, a spoiler leak occurs, and the blocking feature is unavailable at that moment. Stability means that if the connection drops, game/broadcast information or the interpreter crashes, the system should perform a fail-safe action.

The game publishes a minimal deterministic scene description derived from the in-engine state, such as "level", "boss_fight", and a canonical "scene" identifier, with an optional "chapter" when time-based rollouts are

Table 1 Requirements of control protocol

Metric	Definition	Target
L_{e2e} P50/P95/P99	Event→viewer block latency	P95 $\leq$ 500 ms, P99 $\leq$ 2 s
False positives	Blocks in non-spoiler segments(except fall-back block)	0
Partial miss	Exposure before block per event	$\leq$5 s
Availability	Control the success rate during the stream	$\geq$99.9% per hour
Stability	Crashes/drops in 1 h/12 h	0
Security overhead	ΔL with OTP verification	$\leq$30 ms (advised)

used. The broadcaster reports whether a session is live and, for each active channel (streamer), the "platform", "channel_id", and "monetized" status of the streaming. These schemas define the observable surface on which rules apply consistently across builds and pipelines.

Publisher policies are implemented as executable rules. A license script can unlock early chapters after release-offset dates while treating all other content as spoilers by default. It can also target late-game or epilogue scenes when monetized to trigger broadcast blackout and limit gameplay, or relax conditions for specific channels when authorized. Per-scene interaction shaping lets creators continue streaming without revealing sensitive branches; for example, only a spoiler-safe menu option is allowed in a particular scene.

The wire protocol uses typed messages for discovery, license exchange, broadcast checks, state publication, active control, and closure. These include a handshake, license request/response, broadcast check request/response, game and broadcast state messages, and a closure notification. All messages contain UTC timestamps for latency tracking, auditing, and recovery.

Operationally, the system prioritizes blocking spoilers over audiovisual continuity. When the state is missing, delayed, or unverifiable, the default is to block as a fail-safe. However, during connected states, when the disallow conditions are not met, streaming is permitted. Minimum hold-and-release intervals on block states help prevent flickering across adjacent events.

Latency is the total time from the in-engine trigger to the first frame shown to the broadcast SW, and it represents enforcement. The measurement includes event detection, state collection, rule evaluation, command issuance, application in OBS, encoder queue input, and player rendering. Targets are a median close to 200 ms, P95 $\leq$ 500 ms, and P99 $\leq$ 2 s at 1080p60 over consumer Wi-Fi with a capture card.

The communication cadence must maintain the latency budget despite varying network delays. Commands execute atomically, so blackout and mute changes occur with minimal delay. Sessions end with an explicit notification that resets the state and finalizes logs, preventing unnecessary effort to manage when off-streaming.

Security requirements include transport authentication, command integrity, and replay resistance, all achieved without significant latency. Control exchanges utilize mTLS. One-time tokens with nonces link evaluation to the current session within a bounded clock skew; recognition should add $\leq$30 ms. Commands are idempotent and timestamped, ensuring late or duplicate messages cannot reactivate video after a blackout.

Interoperability currently requires working with a game module and an OBS plugin via HTTP. The DSL must be deterministic so that identical states produce identical intents. The interpreter needs to be lightweight enough to run in-process on both sides without noticeable increases in CPU or memory usage.

Human factors are explicit. The system eases streamer cognitive load by automatically enforcing policies at spoiler boundaries, removing the need to memorize changing rules during live play, reducing errors and stress, and maintaining creative flow while compliant content remains on air.

4 Protocol Design

4.1 License-script DSL

Publisher policy is a script that runs and is checked against synchronized game and broadcast states. Rules are deterministic and ordered; ties are resolved by explicit priorities, ensuring the same inputs always produce the same outputs. The interpreter enforces hysteresis by requiring a minimum hold-and-release time of $\geq$500 ms to avoid flickering between neighboring events. Each evaluation records a reason code in the log, helping operators understand why a block or allowance occurs. The DSL is a pure function from "{game, broadcast, env, license.meta}" to "{broadcast_action, game_action, is_spoiler}" that uses fully qualified paths and operators, including equality, set membership, numeric order, and date offsets. The extended policy pack shows monetization-based late-game suppression and channel allowlisting.

For example, the rule "broadcast.is_monetized == true" AND "game.scene $\in$ {late, epilogue}" $\Rightarrow$ "broadcast_action = blackout", "game_action = NotSelection <4>" is encoded as a single clause; another clause allowlists a specific channel ID "AB"; a third unlocks chapters "$\leq$2" after "release_date + 14 days" – time-rollout policy ships separately with a fail-closed default ("blackout" + "block") and staged date gates.

The current syntax of the DSL includes operators like equal/include, numeric operator, and Boolean operators, as well as types such as Boolean, date and time, integer, fractional number, and string, and does not yet include loop/jump instructions.

4.2 State Model

Inputs consist of a single game state call and a broadcast state snapshot as JSON. The game provides a "GameScene" with necessary fields like "level",

"boss_fight", and "scene" (and "chapter" when time rollouts are involved). This removes ambiguity from builds and keeps payloads minimal. The broadcaster sends out "BroadcastStatus" with "is_broadcasting" and an array of "{platform, channel_id, monetized}" entries; the interpreter can offer convenient views such as "broadcast.is_monetized" and "broadcast.channel_ids", while maintaining the raw array for auditing. Outputs include control intents for both sides: the broadcast path applies "{blackout, allow}" now and may extend to "{mute, blur}"; the game path applies "{allow, block}" with interaction shaping such as "Selection<n>" or "NotSelection<...>". Policies may include a TTL.

4.3 Transport and Session

Sessions begin with license discovery and exchange, followed by broadcast checks and periodic state updates, concluding explicitly. Messages are typed and timestamped at the origin. The protocol library defines constructors and validators, enforces idempotency on broadcast actions, and limits enum values (e.g., "action $\in$ {blackout, allow}"; closure "reason $\in$ {game_exit, broadcast_end}"). Timestamps are used to measure latency and recovery.

4.4 Security

All control exchanges utilize mTLS. The license request includes an OTP "auth_token"; verification associates the script with the current session, enforces a nonce to prevent replay attacks, and applies a bounded clock-drift policy. A small cache window can distribute verification costs without exceeding the $\leq$30 ms overhead limit specified in Section 3.

4.5 Failure Handling and Defaults

The protocol is designed to be fail-safe. If the state is missing, late, or cannot be verified, evaluation defaults to the active policy pack. During rollout, that default is "blackout" on the broadcast side and "block" in the game, preventing leakage during stalls, restarts, or desynchronization. In selection-limit packs, the default is permissive because constraints are scoped to named scenes. Receivers treat late or duplicate control messages as idempotent; a stale "allow" cannot re-enable video once a newer "blackout" has taken effect. Retries employ exponential backoff and resync re-establishes the state after transient faults.

4.6 Auditability

Each message includes a UTC timestamp; logs are immutable and can be exported. A trace identifier links event detection, rule evaluation, command emission, and OBS application, helping investigators (primarily IP holders) to reconstruct decisions without reprocessing frames. Control messages display "action" and optional "is_spoiler" for downstream analytics; closure records the final reason and the last channel map to define sessions for auditing and reproducibility. Because of the protocol concept, this log is intended not to supervise streamers but to debug the license script for the IP holder (publisher).

5 Implementation (Figure 1)

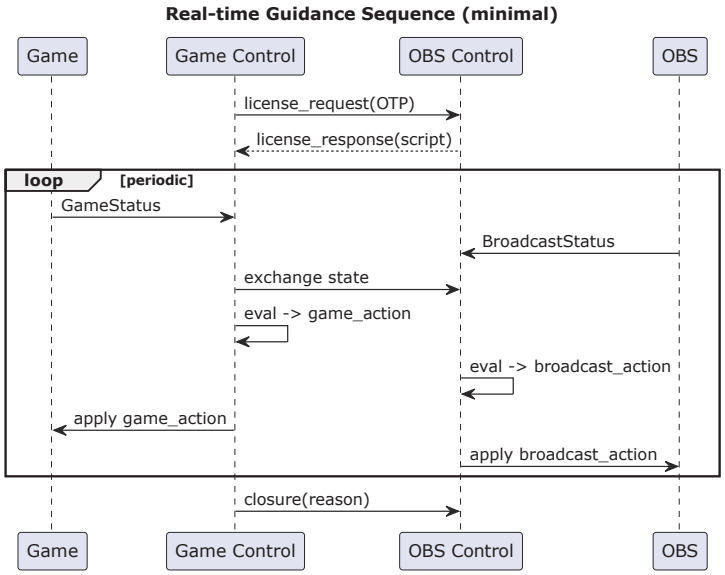


Figure 1 Simplified protocol diagram.

5.1 Game Side

A game control module uses the same license-script interpreter (.dll) as the broadcast SW. It queries a single in-process game-state API and serializes a compact "GameState" JSON object, including elements such as "level", "boss_fight", and "scene". These can vary for each game. Titles

that use time-based rollout also provide or derive a “chapter” and date from the machine so that rules can apply date offsets. The module supplies “{game, broadcast}” state along with the loaded license script to the interpreter. The interpreter is a pure function from “{game, broadcast, env, license.meta}” to “{broadcast_action, game_action, is_spoiler}”, referencing fully qualified paths such as “env.now”, “game.scene”, “game.chapter”, “broadcast.is_broadcasting”, “broadcast.is_monetized”, and aggregates like “broadcast.channel_ids”. The output control value is stored in shared memory or at the process scope and accessed by gameplay code at decision points. If “game_action = block”, revealing interactions are denied. When “Selection<n>“ or “NotSelection<. . . >“ is returned, the UI restricts options in that scene. When “allow” is specified, standard mechanics proceed. The protocol does not specify how a title maps “game_action” to mechanics; each game handles actuation as needed.

The session setup starts on the game side. The game issues a “license_request(game_id, user_id, auth_token)” and receives a script string. The same script loads on both sides to ensure deterministic evaluation. During runtime, the game publishes “game_state” snapshots and adds UTC timestamps for auditing and latency tracking. It also receives “broadcast_state” messages from the broadcaster containing the computed “action” and an optional “is_spoiler”, so both sides evaluate against synchronized inputs. Communication remains continuous, with each side gathering its own state, exchanging snapshots, and assessing the same license.

When using the rollout pack, the control module unlocks early chapters after “release_date + offsets” and defaults to “blackout” on the broadcast side and “block” in the game elsewhere. The selection-limit pack restricts menus during live sessions while keeping the broadcast visible. The extended pack (“rules_extended.json”) includes monetization and channel gating features. For example, if “broadcast.is_monetized = true” and “game.scene $\in$ {late, epilogue}”, then “broadcast_action = blackout” and “game_action = NotSelection<4>“; a whitelist allows channel ID (streamer) “ABC”; a date rule unlocks chapters “$\leq$2” after “+14 days”. These policies require no binary change; the interpreter executes the script as is.

5.2 Broadcast Side

An OBS plugin embeds the interpreter and provides an HTTP API to the protocol peer over mutual TLS (mTLS) on the Game side. It maintains a process-local “BroadcastStatus” with “is_broadcasting” and an array

of "{platform, channel_id, monetized}" tuples. When "broadcast_action = blackout", the plugin applies a preconfigured scene or filter stack to hide the video, and can also mute the audio. When "allow", it restores the program chain. Future policies may add "mute" or "blur" as primary actions. Commands are idempotent and timestamped. A late or duplicate "allow" cannot re-enable video once a newer "blackout" is active. A shared validator enforces that on-wire enums include only "{blackout, allow}".

The broadcaster responds to "broadcast_check_request" with "is_broadcasting" and "broadcasting_info", emits "broadcast_state" with the computed "{action, is_spoiler}", and logs reason codes. It verifies the incoming "auth_token" using OTP with nonce and bounded skew. It accepts the state and applies actions atomically, ensuring blackout and mute occur within the same frame window. The plugin publishes UTC-stamped messages and retains trace IDs to correlate evaluation, command emission, and OBS application.

5.3 Deployment Notes

The setup uses two independent machines connected to the same consumer Wi-Fi access point, along with an HDMI capture card at 1080p resolution at 60 frames per second. Transport employs authenticated HTTP with mutual TLS. The OTP and nonce link the license to the active session. Security details beyond OTP and mTLS are reserved for separate work. Failure handling is a fail-safe. If the state is missing, late, or cannot be verified, the policy in effect at the time is used by default. During rollout mode, the default is "blackout" and "block" with "is_spoiler = true", preventing leaks during stalls, restarts, or clock skew. In selection-limit mode, the default is permissive because constraints apply only to specific scenes. Resync restores the state after temporary faults. All decisions are logged in shared records with timestamps and reasons for dispute resolution and reproducibility.

6 Experimental Methodology

6.1 Testbed

The evaluation tests whether a license-script protocol can suppress spoiler spans in real-time on a typical creator setup, while maintaining permitted content and easing the streamer's workload. Two independent PCs share a single consumer Wi-Fi access point. The Game PC runs the in-process control module and embedded interpreter, timestamping events as they occur. The

Broadcast PC uses OBS with an HDMI capture card to capture 1080p60 video. Both sides provide standard state contracts – "GameScene" on the game side and "BroadcastStatus" on the broadcast side – and communicate via typed messages over a secure HTTP connection with mTLS and session-bound OTP. This setup mirrors the reference compliance flow used for discovery, mutual verification, rule evaluation, active control, and clean shutdown.

6.2 Instrumentation

Instrumentation follows a fixed timeline from in-engine trigger to the first viewer frame that reflects enforcement: T_0 event, T_1 state collection, T_2 rule-evaluation start/end inside the interpreter, T_3 command issue, T_4 OBS apply, T_5 encoder enqueue, and T_6 player render. The game overlays a millisecond burn-in timer that encodes T_0 and T_3; the receiver extracts T_6 via frame-accurate OCR. All protocol messages carry UTC timestamps and a trace ID, allowing logs from both processes to correlate without requiring video decoding. Clock offset is measured and corrected at session start to bound timestamp skew.

6.3 Scenarios

Scenarios include nominal operation, spoiler control under load, fault and recovery, and cryptographic overhead. A 10-minute non-spoiler segment verifies zero false positives. Thirty spoiler episodes of 30–600 s test start and end boundaries and adjacency; a minimum of 1 s separation ensures hysteresis prevents visible flicker. Fault injection drops the state stream for 1 s and 3 s, restarts the OBS plugin mid-session, and induces transient Wi-Fi RTT spikes; a complete disconnect and reconnect confirm that the fail-safe default (block) prevents leakage during impaired links. Security trials toggle OTP validation, test valid, expired, and rejected tokens, and vary cache windows to isolate verification costs. Duration runs include a one-hour baseline and a long soak of up to 12 hours to evaluate rule toggling stability, CPU/memory usage, and sustained availability.

6.4 Procedure

1. Compute segment latencies $L_i = \Delta(T_i, T_{i+1})$ and end-to-end latency $L_{e2e} = \Delta(T_0, T_6)$; report P50/P95/P99 and jitter.
2. Perform overlap analysis: calculate the IoU between labeled spoiler intervals and enforced block windows, and sum the leakage per event.

3. Check false positives: non-spoiler segments must show zero block toggles.
4. Measure failure recovery: time to safe block on fault and time to steady state after recovery.
5. Measure security overhead: ΔL with OTP on/off, failure-path latency, and retry−policy impact.

Metrics measure latency, accuracy, stability, and security overhead against specific targets. Acceptance requires a median near 200 ms, with P95 $\leq$ 500 ms and P99 $\leq$ 2 s at 1080p60 over consumer Wi-Fi with capture. Accuracy demands zero false positives and $\leq$5 s of total pre-block exposure per event. Stability is summarized as hourly enforcement availability with MTTR for injected faults and no crashes or drops during the one-hour baseline. Security overhead is the extra ΔL with OTP enabled, maintaining a steady-state budget of $\leq$30 ms.

Analysis proceeds reproducibly from synchronized logs and captured video. For each event, the evaluator calculates all L_i and L_{e2e}, aggregates distributions, and generates a CDF for L_{e2e} along with box plots for each L_i. Time-aligned timelines display interpreter decisions, broadcast actions, and spoiler labels to highlight any flicker or overlap mistakes. Resource traces from both processes measure interpreter and I/O overhead. Security trials report verification latency with and without caching, as well as the behavior of replay resistance under nonces and bounded clock skew. When relevant, results are compared with console-level blocking and viewer-side filters to show the effect of policy enforcement at the source.

Human-factor observations accompany technical results. Since the interpreter automatically applies policy at boundaries, streamers do not need to memorize evolving guidelines during live play. Errors and stress decrease while creative flow remains intact. This advantage is inherent to contract-driven automation and is a primary reason to favor protocol enforcement over static documents or post-hoc moderation.

7 Results (Table 2)

- Latency: mean $\approx$65 ms; P95 $<$ 75 ms; P99 $<$ 170 ms.
- Accuracy: 0 false positives; leakage $\leq$5 s/event.
- Stability: availability $\geq$99.9%/h; no crashes/drops in 1 h; MTTR reported.
- Overhead: small CPU/memory deltas; ΔL with OTP within budget.

Table 2 Latency and accuracy

Condition	L_{e2e} P50	P95	P99	Approx. CPU Overhead	False Positive
Congested Wi-Fi	52 ms	72 ms	137 ms	0.21%	0
OTP on	60 ms	69 ms	166 ms	0.76%	0

- Communication frequency vs. load: monotonic load increase, with no saturation observed in the tested range.

The protocol met the latency targets within the defined test environment. During over one-hour tests at 1080p60 on consumer Wi-Fi with an HDMI capture path, end-to-end block latency averaged around 65 ms, with P95 $\leq$ 75 ms and P99 $\leq$ 170 ms from the in-engine trigger to the first viewer frame, indicating enforcement. Measurements follow the $\mathrm{T}_0 \rightarrow \mathrm{T}_6$ path, using timestamped wire messages and logs for correlation.

Accuracy met the acceptance criteria. Non-spoiler intervals showed no false positives. For labeled spoiler episodes lasting 30–600 s, the cumulative pre-block exposure per event remained within the $\leq$5 s limit. The intersection-over-union between labeled spoiler spans and enforced block windows was high across all scenarios, including adjacent events separated by 1 s, where hysteresis prevented visible flicker.

Stability remained within requirements. Hour-long sessions sustained $\geq$99.9% enforcement availability without crashes or drops. Recovery from injected state drops, plugin restarts, and transient Wi-Fi RTT spikes maintained the fail-safe posture until normal signaling resumed, then returned to steady operation without manual intervention. Clean termination generated a closure record with terminal reason and last channel map, ensuring audit completeness per session.

Security features did not significantly impact latency. Enabling OTP during license exchange with nonces and a bounded clock skew added only negligible ΔL within the 30 ms advisory limit. The authenticated "auth_token" tied evaluations to the active session, thereby preventing replay attacks.

Overhead aligned with real-time use. Publishing the minimal canonical game state "{level, boss_fight, scene}" and broadcasting status "{is_broadcasting, broadcasting_info[]}" kept payloads small and validation predictable, reducing interpreter costs and preventing measurable CPU or memory growth compared to the baseline OBS capture. Varying communication frequency produced a smooth load curve with no saturation at the tested cadence, thus maintaining the latency budget.

Policy variants performed as expected. The rollout-by-date pack defaulted to spoiler-safe blackout and in-game block until early chapters unlocked the time gates, confirming the fail-safe baseline under link impairment. The selection-limit pack restricted in-scene choices during live sessions, while the broadcast remained visible, demonstrating that interaction shaping can prevent narrative leaks without hiding video. The extended pack enforced monetization-based late-game suppression and adhered to explicit channel allowlists, showing that business rules can coexist with narrative rules.

Human-factor observations aligned with the design goal. Because the interpreter automatically enforced the license at spoiler boundaries, streamers didn't pause to review per-scene rules or manage hotkeys. Cognitive load and stress decreased while the permitted content remained on air.

All results were produced using the shared wire model and helpers, including typed messages for discovery, license exchange, broadcast checks, state publication, active control, and closure, each stamped at the source for audit and reproducibility.

8 Limitations and Future Work

The method relies on accurate and timely state information from the game and the broadcaster. Missing fields or mislabels can cause misses or over-blocks. The interpreter is deterministic, so bad inputs produce incorrect but repeatable actions. The model does not detect spoilers on its own. It executes the license. Therefore, rule authoring quality and test coverage are essential. Ultra-low-latency modes reduce the $\mathrm{T}_3 \rightarrow \mathrm{T}_6$ budget and stress encoder and player queues. Wi-Fi variability and capture hardware introduce fixed and random delays. These delays dominate the tail and lie outside the interpreter.

Future work involves four main areas. First, transport and timing: transition control to WebSocket, incorporate pacing with back-pressure, and improve clock synchronization. Predictive holds near known boundaries can smooth short bursts while avoiding false positives. Second, rule creation and distribution: deliver rules as signed artifacts with semantic versioning and staged rollout, supply them from a PDS or license vault, and include dry-run validation that replays recent logs before activation. Third, coverage and scaling: expand to hardware switchers and multi-input mixers, implement multi-instance orchestration, and standardize the "GameScene" and "BroadcastStatus" contracts for cross-title reuse with public schema and conformance tests. Fourth, entitlement and platform collaboration: combine source-side blackouts with viewer gating so purchasers see the full range.

Conversely, others see safe placeholders as a way to shift audience checks to platform edges when possible..

Security can be improved through key rotation, shorter OTP TTLs, and formal replay proofs. UX needs clear on-screen notices during blackouts and public reasons in logs so disputes can be resolved without needing code access. The goal is a standard, not a silo.

9 Conclusion

A license-script protocol can prevent spoiler exposure in live game streams with sub-second P95 latency and zero false positives under typical conditions. It coordinates the game and the broadcaster using minimal state and deterministic rules. It blocks only what the license marks as sensitive while keeping permitted content live. It is not DRM, nor is it post-hoc moderation. It provides real-time, content-conditioned enforcement. Automation reduces the streamer's workload because the interpreter applies policy at boundaries. Creators do not need to memorize evolving rules or juggle hotkeys. Publishers see guidance executed precisely at narrative borders. The reference implementation and evaluation demonstrate feasibility and low overhead. Next steps include standardizing schemas and rules, strengthening distribution and audit processes, and adding optional audience gating that extends protection beyond the source while maintaining lawful access.

References

[1] Atlus, "Persona 5 Royal Video & Livestreaming Guideline", Atlus, 2020. Available: https://atlus.com/persona-5-royal-video-livestreaming-guidelines.

[2] Lee, S., Lee, S., & Baek, H., "How does live streaming impact media content consumption?", Entertainment Computing, 52, 2025.

[3] Twitch, "DMCA Guideline", Twitch. Available: https://legal.twitch.com/legal/dmca-guidelines.

[4] Content Overseas Distribution Association, "Uploader of "gameplay video" found guilty of violating guideline", Content Overseas Distribution Association, 2023. Available: https://coda-cj.jp/en/news/361/.

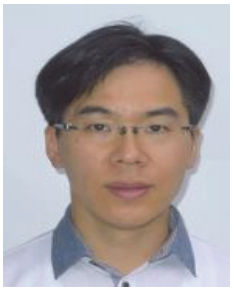
[5] Holden, J. & Schuster, M., "Copyright and Joint Authorship as a Disruption of the Video Game Streaming Industry", Columbia Business Law Review, 2020(3), 2021.

[6] Sony Interactive Entertainment LLC., "Spoiler block service", US12011664B2, United States Patent Office, 2023.
[7] Sony Interactive Entertainment LLC., "Spoiler block service", US11318388B2, United States Patent Office, 2022.
[8] Square Enix, "OCTOPATH TRAVELER II Guidelines for livestreaming and posting videos/images", Square Enix, 2023. Available: https://www.square-enix-games.com/en_US/documents/octopath-traveler-ii-guidelines-for-livestreaming-and-posting-videos-images.
[9] Nintendo, "Nintendo Game Content Guidelines for Online Video & Image Sharing Platform", Nintendo, 2020. Available: https://www.nintendo.co.jp/networkservice_guideline/en.
[10] Atlus, "【2024/3/1更新】『ペルソナ５ ザ・ロイヤル』リマスター版 動画・生放送等配信ガイドラインのお知らせ", Atlus, 2023. Available: https://p-ch.jp/news/8908/.
[11] MiniTool, "Fixed: Gameplay Recording Paused Because You Entered a Blocked Scene", MiniTool, 2021. Available: https://videoconvert.minitool.com/news/gameplay-recording-paused-blocked-scene.html.
[12] Wan, M., Misra, R., Nakashole, N. and McAuley, J., "Fine-Grained Spoiler Detection from Large-Scale Review Corpora", Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, 2605–2610, 2019.
[13] Mäkelä, V., Chinda, K. and Khan, I., "Spoiler Alert! Understanding and Designing for Spoilers in Social Media", ACM IMX '25, 230–242, 2025.
[14] Sony, "How to capture gameplay and screenshots on PS5 console", Sony, Available: https://www.playstation.com/en-us/support/games/capture-ps5-gameplay-screenshots/.

Biographies

YongJoon Joe received his B.A. and M.Sc. in Information Science from Kyushu University, Japan, in 2011 and 2013, respectively. Since 2016, he has been a Director at LSware Inc. (Korea), where he leads R&D on trackable, reliable systems, including blockchain-based Software Bills of Materials (SBoM) for vulnerability and copyright management. His expertise encompasses blockchain, concurrency, security, and copyright, and his research interests extend to game theory and parallel simulation computing.

Sung-Il Jang received his master's degree in Computer Science from Soongsil University in 2019. He is currently a Principal Research Engineer at LSware Inc. in the Republic of Korea. His research areas include distributed computing, blockchain, artificial intelligence, and retrieval-augmented generation (RAG).



DongMyung Shin received his Ph.D. in Computer Engineering from Daejeon University in 2003. Since 2016, he has served as Head of the R&D Division and Senior Managing Director at LSware Inc., Republic of Korea. His research areas include open source licenses, copyright technology, system/network security, software vulnerability analysis and evaluation, and blockchain technology.