VAST XML Validation at Bid-Time Scale: Latency Analysis and Integration Patterns for Programmatic Video Pipelines

Aleksander Sekowski

Independent Researcher

vastlint.org . github.com/aleksUIX/vastlint

Preprint -- April 2026

Abstract

Malformed VAST XML tags are a well-documented but structurally unaddressed source of impression loss in programmatic video advertising. The OpenRTB 2.6 specification defines the protocol for real-time bidding but imposes no requirement on the structural validity of the VAST markup delivered in the bid[].adm field. Industry data suggests 25 to 40 percent of served video impressions fail to render, with structural VAST errors as a primary causal factor. We describe vastlint, an open-source VAST linter with 108 validation rules across VAST 2.0 through 4.3, and report benchmark results measuring per-bid validation latency and system throughput on two representative tag sizes (17 KB and 44 KB). A 17 KB CTV tag validates in 350 microseconds per bid -- 0.35 percent of a 100 ms bid window. A 44 KB tag validates in 2.1 milliseconds -- within the 5 to 10 ms creative selection budget. On 10 cores, the system sustains 16,236 validations per second, exceeding the throughput requirements of a mid-tier exchange on a single instance.

We compare against XSD schema validation using the IAB's own VAST 4.2 XSD and show that while XSD validation is faster (72 us per tag), its findings consist entirely of false positives -- element ordering violations and VAST macro rejection -- that do not correspond to playback failures. We describe four integration patterns (SSP bid response, DSP creative ingestion, SSAI stitch-time, and async observability sidecar) and analyze the economic case for inline validation at each stage. Structural VAST validation at bid-time scale is computationally feasible with negligible latency impact. The absence of such validation in current pipeline implementations represents a recoverable source of revenue loss.

1. Introduction

Programmatic video advertising runs through real-time bidding (RTB) protocols in which a publisher's supply-side platform (SSP) broadcasts an impression opportunity to demand-side platforms (DSPs) [13, 15]. Each DSP evaluates the opportunity and responds with a bid price and ad markup within a time window defined by the tmax field in the OpenRTB specification. For server-side exchanges that window typically ranges from 80 to 1,000 ms [1].

The ad markup for video inventory is a VAST (Video Ad Serving Template) XML document, delivered in the bid[].adm field of the OpenRTB bid response [2]. VAST is an IAB Tech Lab standard covering six major versions (2.0 through 4.3). XSD schemas are published for versions

2.0.1 through 4.2. Version 4.3 has a prose-only specification [3].

When the winning bid carries a malformed VAST tag, the ad fails to render. The impression slot is consumed. The publisher's ad server records a served code but no confirmed impression. The DSP records a win notice but no confirmed delivery. Both parties are technically correct from their own measurement perspective. The revenue disappears in the discrepancy.

Penthera's 2023 research puts VOD ad failure rates at up to 40 percent [4]. Google Ad Manager documents a 25 percent gap between served codes and confirmed impressions as a systemic condition [5]. The IAB's own VAST specification ships with no reference validator and no conformance test suite. Every player, ad server, and SSAI stitcher interprets the spec differently. Structural problems in the markup that are detectable before the tag is committed to the bid response are typically discovered weeks later in reconciliation reports.

The central question is whether structural VAST validation can run inline, within the constraints of a real-time bid pipeline, without material latency cost. We present benchmark data showing it can, describe four integration points where it changes the economics, and release the tool as open-source software.

2. Background and Related Work

2.1 OpenRTB and the Creative Validation Gap

The OpenRTB 2.6 specification [2] defines the protocol for real-time bidding, covering publisher context, device signals, user identity, floor pricing, auction type, and regulatory compliance. The tmax field constrains bid response latency at the exchange level. Google Authorized Buyers, the largest programmatic exchange, requires that 85 percent of bid responses arrive within the stated tmax deadline and documents that tmax typically ranges from 80 to 1,000 ms [1].

The specification is silent on the structural validity of the ad markup in bid[].adm. There is no normative requirement, no optional validation handshake, and no error reporting mechanism for malformed creatives. The protocol assumes the DSP is responsible for the validity of its own markup.

2.2 VAST Specification Coverage

The VAST specification [3] has evolved from version 2.0 (2008) to 4.3 (2022). XSD schemas are available for versions 2.0.1 through 4.2. Those schemas cover structural validity: required elements, element cardinality, and attribute types [14]. They do not cover:

- URI syntax validation (RFC 3986 [6]) on URL fields
- MIME type validation (IANA registry [7]) on MediaFile and resource elements
- Currency code validation (ISO 4217 [8]) on Pricing elements
- Ad identifier format validation (Ad-ID registry [9]) on UniversalAdId elements
- Semantic deprecation rules (e.g., VPAID deprecated in VAST 4.1+)
- Version consistency (declaring version="2.0" while using 4.x elements)

- Cross-field consistency (e.g., Wrapper tags without VASTAdTagURI)

Most production VAST failures are not caused by malformed XML. They are caused by values that are structurally valid XML but semantically wrong. A MediaFile MIME type that matches no known codec. An HTTP media URL on HTTPS inventory. A VPAID declaration on a CTV endpoint that silently filters it. None of these fail an XSD schema check.

2.3 Existing Validation Tools

The IAB Tech Lab does not publish a reference VAST validator. Existing tools include:

- VAST Inspector (Google): A manual web-based debugging tool. Not embeddable, not programmatic, no batch support.
- AdTechTesting validators: Web-based inspection tools for manual QA workflows.
- XSD validation: Well-supported in Java, .NET, and Python ecosystems. Covers structural conformance to a schema but not the semantic gap described in Section 2.2. We benchmark XSD validation directly in Section 5.4 and find that the IAB's own VAST 4.2 XSD cannot validate production VAST tags without modification, and when adjusted for a fair comparison, produces only false positives on well-formed tags.

We are not aware of prior published work benchmarking VAST validation latency against real-time bid pipeline constraints or describing a validation library designed for inline bid-time use.

3. vastlint: Design and Rule Coverage

vastlint is a VAST XML linter written in Rust. It exposes bindings for Go (via CGo), WebAssembly (npm package), and a command-line interface. The core library has three runtime dependencies: quick-xml for XML parsing, url for URI parsing, and phf for compile-time hash maps.

3.1 Rule Coverage

108 validation rules across VAST 2.0 through 4.3, organized into six categories:

1. Schema compliance: Required elements, element cardinality, attribute presence and type, derived from IAB XSD schemas (2.0.1 through 4.2) and spec prose (4.3).
2. URI validation: RFC 3986 syntax on every URL field (Impression, TrackingEvent, MediaFile, VASTAdTagURI, ClickThrough, and others).
3. MIME type validation: IANA registry lookup on MediaFile type and resource element type attributes.
4. Currency validation: ISO 4217 lookup on Pricing currency attributes.
5. Ad identifier validation: Ad-ID registry format on UniversalAdId values.
6. Semantic and deprecation rules: VPAID deprecation in VAST 4.1+, HTTP URLs on HTTPS inventory, version and element consistency, duplicate tracking pixels, Wrapper chain integrity.

Each finding includes a rule ID, severity (error, warning, or info), XPath location within the document, a human-readable message, and a spec reference such as "IAB VAST 4.2 S3.4.1".

3.2 Version Coverage

VAST Version	XSD Available	Schema Source	Notes
2.0 / 2.0.1	Yes	IAB Tech Lab	Basic inline and wrapper
3.0	Yes	IAB Tech Lab	Non-linear ads, icon support
4.0	Yes	IAB Tech Lab	UniversalAdId introduced (required)
4.1	Yes	IAB Tech Lab	VPAID deprecated
4.2	Yes	IAB Tech Lab	Verification, macro improvements
4.3	No (prose only)	IAB spec prose	Interactive ads, most recent version

3.3 Implementation

The parser performs a single-pass DOM traversal of the VAST XML document. Rules apply per-element during traversal with no backtracking. Lookup tables for IANA types, ISO currency codes, and Ad-ID patterns are compiled into the binary via phf with zero runtime allocation for table construction. The Go and WASM bindings are thin FFI wrappers around the same compiled rule set.

4. Methodology

4.1 Test Corpus

Two VAST corpora were constructed to bracket the size range observed in production CTV ad serving:

- 17 KB corpus: A standard CTV inline tag with impression tracking, companion ads, three to four MediaFile renditions, and click tracking. Common in mid-tier CTV inventory.
- 44 KB corpus: A heavy CTV tag with extensive tracking pixels, multiple companion sets, many MediaFile renditions, and several impression pixels. Common in premium and network CTV.

Corpus templates are based on de-identified production VAST tags from CTV ad serving workflows. URLs, tracking domains, creative identifiers, and other fields that could identify specific demand partners or publishers were replaced with synthetic values. Tag structure, element nesting, attribute patterns, and field counts reflect real-world usage. Field content was varied programmatically across corpus files to reduce branch prediction artifacts. Both corpora and the generation scripts are available in the project repository.

4.2 Benchmark Configuration

All benchmarks ran on an Apple M4 (10-core, 16 GB RAM) in a controlled single-session test with no concurrent workloads. Each benchmark ran a minimum of 100 iterations with a warmup period. Results represent mean latency across all iterations.

Runtime versions:

- Rust: 1.87.0 (release build, opt-level = 3)
- Go: 1.24.1 (CGo enabled, release build)
- Node.js: 22.11.0 (WASM via @wasmer/wasi)
- Python/lxml: 3.13 / lxml 6.0.4 (libxml2, used for XSD comparison)

Multi-core benchmarks used Rust's rayon crate for work-stealing parallelism across 10 threads, Go's goroutines across 10 goroutines, and Python multiprocessing across 10 processes for XSD.

4.3 Pipeline Context

Benchmark latencies were evaluated against the following pipeline time budgets, drawn from industry sources:

Pipeline stage	Time budget	Source
Full RTB auction (end-to-end)	150 ms	AdtechSchool [10], industry standard
DSP decision pipeline	40 to 80 ms	AdtechSchool [10]
DSP creative selection (stage 4)	5 to 10 ms	AdtechSchool [10]
SSP bid response window (server-side)	80 to 150 ms	OpenRTB 2.6 [2], Google [1]
SSAI stitch window	50 to 200 ms	Practitioner range
OpenRTB tmax (Google range)	80 to 1,000 ms	Google Authorized Buyers [1]

5. Results

Two metrics matter for evaluating inline validation feasibility:

- Per-bid latency: The time a single bid response is delayed by validation. In a typical SSP architecture, each bid is processed on one thread (or one goroutine/async task). Validation runs synchronously within that context. This is the latency the bid actually waits for.
- System throughput: The total number of tags per second the validation layer can sustain across all cores. This determines how large an exchange a single instance can serve.

These are distinct metrics. Per-bid latency governs whether validation fits within a single bid's time budget. System throughput governs infrastructure sizing.

5.1 Per-Bid Validation Latency

Each tag is validated on a single thread. All measurements are 100,000 iterations on Apple M4.

Tag size	Per-bid latency	% of 100 ms bid window	% of 5-10 ms creative selection budget
17 KB	**350 us**	**0.35%**	3.5-7%
44 KB	**2,142 us**	**2.14%**	21-43%

A 17 KB tag adds 350 microseconds to the bid response path -- 0.35 percent of a 100 ms tmax window. This is smaller than the variance in a typical network hop and well below the creative selection stage (5 to 10 ms) that already runs on every bid.

A 44 KB tag adds 2.1 milliseconds. This is larger but still within the creative selection budget at most exchanges. For pipelines where 2 ms of inline latency is unacceptable, the async sidecar pattern (Section 6.4) provides the same validation without blocking the bid path.

5.2 System Throughput (10 Cores)

When processing a concurrent workload across 10 cores, throughput determines how many bids per second the validation layer can handle.

Tag size	Throughput (10 cores)	Infrastructure implication
17 KB	**16,236 tags/sec**	Single instance serves a mid-tier exchange
44 KB	**2,566 tags/sec**	Single instance serves ~220M tags/day

A mid-tier SSP processing 10,000 bid responses per second at 17 KB requires a single 10-core instance with headroom to spare. At 44 KB, the same instance handles 2,566 tags per second, sufficient for all but the largest exchanges, and horizontally scalable beyond that.

5.3 Cross-Runtime Performance

vastlint exposes the same rule engine through multiple runtime bindings. The table below reports per-bid latency (single-thread) and system throughput (10 cores) for each runtime.

Runtime	17 KB per-bid	44 KB per-bid	17 KB throughput (10 cores)	44 KB throughput (10 cores)
Rust (core)	**350 us**	**2,142 us**	**16,236/s**	**2,566/s**
Go (CGo)	403 us	2,269 us	13,558/s	1,993/s
WASM (Node.js)*	483 us	2,725 us	--	--

WASM runs in a single-threaded V8 isolate. Multi-core throughput is not applicable. WASM deployments (edge functions, browser extensions) are single-request contexts where per-bid latency is the relevant metric.

The Go bindings add roughly 15 percent overhead on per-bid latency and reach 84 percent of Rust throughput on 17 KB tags at 10 cores. The CGo FFI boundary does not change the deployment calculus.

5.4 Comparison with XSD Schema Validation

To establish a baseline, we benchmarked the IAB's own VAST 4.2 XSD schema [3] using lxml 6.0.4 (backed by libxml2) on the same corpus and hardware. XSD validation is the closest existing approach to structural VAST validation and is well-supported across production runtimes.

XSD setup and fairness adjustments. The IAB VAST 4.2 XSD declares targetNamespace="http://www.iab.com/VAST", but no production VAST tag in the wild uses that namespace -- tags use a bare <VAST> root element without namespace qualification. When validated as-is, lxml rejects the root element on line 2 and returns a single error without inspecting the rest of the document. To ensure a fair comparison where the XSD validator walks the full document tree, we stripped the target namespace from the XSD and reordered corpus elements in

memory to satisfy xs:sequence ordering constraints. After these adjustments, lxml's deepest errors reached line 185 of 212 (87.3%) on the 17 KB tag and line 467 of 494 (94.5%) on the 44 KB tag, confirming near-complete tree traversal.

Both benchmarks ran 100,000 iterations on the same files using the same hardware (Apple M4). Per-bid (single-thread) latency reported for apples-to-apples comparison.

Tool	17 KB per-bid	44 KB per-bid	Rules	Findings (17 KB)	Findings (44 KB)
lxml XSD (libxml2)	**72 us**	**169 us**	XSD structural	5	10
vastlint (Rust)	350 us	2,142 us	108 semantic	3	3

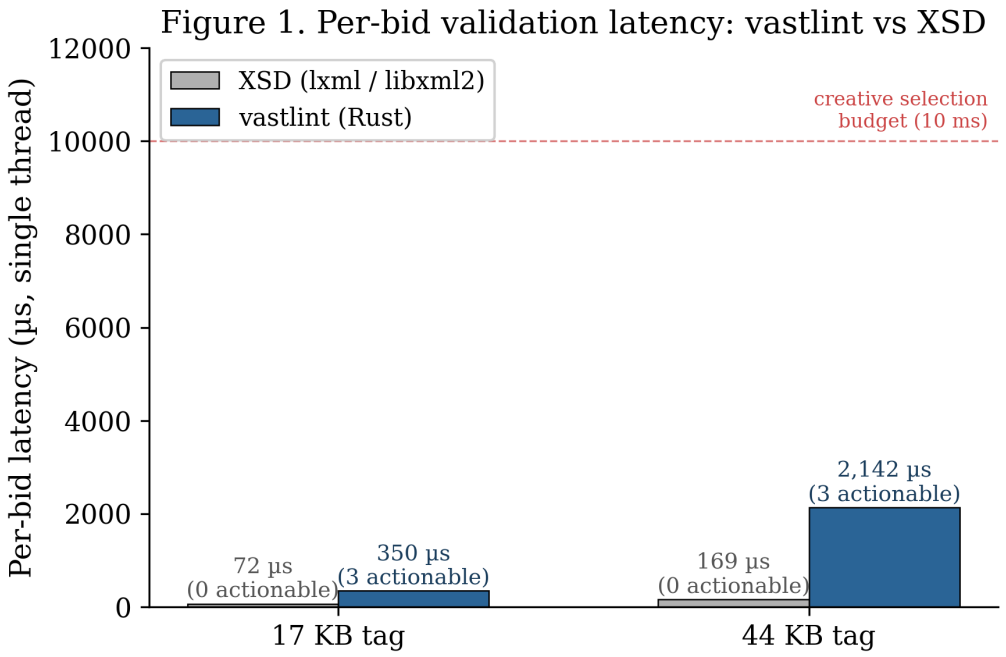

Figure 1. Per-bid validation latency: vastlint vs XSD. Both bars fit below the 10 ms creative selection budget. XSD findings (gray) are entirely false positives; vastlint findings (blue) are actionable.

XSD validation is faster: roughly 5x on 17 KB and 13x on 44 KB. This is expected: libxml2 is a mature C library performing a single-pass schema check, while vastlint applies its full rule set including URI parsing, MIME type lookup, currency validation, and cross-field consistency checks.

However, the two tools are not solving the same problem. The XSD's 5 findings on the 17 KB tag were:

1. Element ordering violations (3 of 5): The XSD uses xs:sequence, which requires child elements in a fixed order. Production VAST tags routinely place elements in non-schema order. A <Creatives> block appearing before <Impression> is flagged as an error even though every VAST player and ad server processes it correctly regardless of order.
2. VAST macro rejection (2 of 5): URL fields containing VAST macros like [ERRORCODE] or [CACHEBUSTING] are rejected as invalid xs:anyURI. These macros are required by the VAST specification itself -- they are placeholder tokens that the video player substitutes at runtime. Rejecting them is a false positive.

The XSD's 10 findings on the 44 KB tag follow the same pattern: element ordering and macro rejection, with additional CompanionAds ordering violations.

vastlint's 3 findings on both tags were:

1. VAST-2.0-version-mismatch (error): The tag declares version="2.0" but uses elements introduced in later spec versions.
2. VAST-3.0-skip-event-no-skipoffset (warning): A skip tracking event is declared without a corresponding skipoffset attribute, meaning the event can never fire.
3. VAST-4.1-tracking-event-value (warning): A tracking event uses a value not defined in the VAST 4.1 tracking event vocabulary.

All three vastlint findings identify conditions that affect ad playback or measurement. None of the XSD findings do. The XSD is faster because it does less meaningful work: it checks structural conformance to a schema that does not model real-world VAST usage. The IAB's own XSD cannot validate a production VAST tag without modifications, and when it does validate, it flags spec-compliant constructs as errors.

The relevant question is not which tool is faster, but which tool's findings are actionable at bid time. A validator that runs in 72 microseconds but produces only false positives does not improve pipeline economics. vastlint at 350 us per bid on a 17 KB tag -- 0.35 percent of a 100 ms bid window -- buys a rule set that catches errors players and ad servers actually fail on. Even at 2.1 ms on a 44 KB tag, the cost fits within the creative selection budget (5 to 10 ms).

6. Integration Points

The following integration patterns are based on the benchmark data and the pipeline stage analysis in Section 4.3. None have been deployed in production at the time of writing.

6.1 SSP Bid Response Validation

The highest-value integration point is the SSP validating bid[].adm before committing to the bid response. A malformed tag that wins the auction consumes the impression slot, charges the publisher's fill rate, and sinks the bid cost. No rendered ad. Validation before the response is serialized enables two recovery paths: runner-up substitution (awarding the slot to the next-highest bid) or no-fill return (falling through to the publisher's next demand source). Neither path is available after the response is sent.

Cost: 350 us per bid on a 17 KB tag (0.35% of a 100 ms window). 2.1 ms on a 44 KB tag (within the 5-10 ms creative selection budget). System throughput: 16,236 tags/sec on 10 cores.

6.2 DSP Creative Ingestion

Validation at creative upload is not subject to real-time constraints. Catching structural errors at ingestion (a missing <Duration>, an absent <MediaFile>, a VPAID declaration on VAST 4.1+) prevents broken tags from entering rotation for the life of the campaign.

Cost: One validation per creative upload. Latency is irrelevant.

6.3 SSAI Stitch-Time Validation

Server-side ad insertion stitchers receive VAST responses and transcode them into the content stream. A malformed tag at stitch time produces a visible blank in the viewer's stream with no retry. Validation before stitching enables a backfill decision within the stitch window (50 to 200 ms). Even the 44 KB per-bid cost (2.1 ms) fits within that window with room to spare.

6.4 Async Observability Sidecar

For pipelines where inline validation is not feasible, a sidecar process can validate asynchronously and feed results into an observability stack. A single 10-core instance handles 16,236 tags per second at 17 KB. Bad tags are not blocked from serving, but creative error rates become visible in real time by demand partner.

7. Economic Analysis

The economic case for inline validation rests on an asymmetry between validation cost and recoverable revenue. On the cost side, validating 10,000 tags per second at 17 KB requires a single 10-core instance (16,236 tags/sec). On cloud infrastructure, that is single-digit dollars per day in compute.

On the revenue side, consider an SSP processing 10,000 bid responses per second. Industry data suggests tag failure rates between 8 and 40 percent (Section 1), though the upper range includes network failures, player bugs, and rendering issues outside the scope of tag linting. A conservative estimate of the structurally detectable fraction (missing required elements, bad URL schemes, deprecated API declarations, invalid MIME types) is 20 to 40 percent of total failures. At CTV CPMs ($10 to $25), even a small fraction of recovered impressions dwarfs the infrastructure cost by orders of magnitude.

The actual dollar figure depends on the exchange's partner mix, tag quality distribution, and how aggressively malformed tags are already filtered by other means. We do not have production deployment data to narrow these ranges. The claim is not a specific dollar figure but the structural observation: when validation costs single-digit dollars per day and serves millions of impressions, the math works under any reasonable failure rate.

8. Discussion

8.1 The OpenRTB Creative Validation Gap

OpenRTB enforces bid response latency through tmax and exchange-level filtering but has no mechanism for creative quality enforcement at the protocol layer. The protocol is not in the business of ad markup semantics. But that means validation is entirely the responsibility of individual bidders, and in practice it is rarely done inline.

Structural VAST failures propagate through the full pipeline before anyone detects them. By the time a malformed tag surfaces in reconciliation data, the auction has occurred, the publisher's

inventory has been consumed, and the revenue cannot be recovered.

The benchmark data shows inline validation does not require a latency tradeoff. Per-bid cost ranges from 350 us (17 KB) to 2.1 ms (44 KB), fitting within existing pipeline budgets. System throughput on 10 cores exceeds the requirements of a mid-tier exchange. Bindings exist for every major runtime. The XSD comparison in Section 5.4 shows that existing schema-based approaches, while faster, do not address the semantic errors that cause playback failures -- making raw speed the wrong metric for this problem.

8.2 Limitations

All benchmarks ran on a single hardware configuration (Apple M4, 10-core, 16 GB RAM). Results on x86-64, ARM server, or cloud VM architectures will differ.

The corpora are synthetic: structurally modeled on de-identified production CTV tags, with programmatically varied field content. They cover the element nesting, attribute patterns, and size ranges common in mid-tier and premium CTV inventory. They do not capture the long tail of malformed tags that account for the highest failure rates in production -- deeply nested wrappers, truncated XML from timeout conditions, binary-encoded companion assets, or tags that mix elements from multiple VAST versions in non-standard ways. Validation latency on pathological tags may differ from the numbers reported here.

The 17 KB and 44 KB sizes were selected based on the author's experience with CTV ad serving. Corpus generation scripts and de-identified templates are in the repository to allow reproduction on different hardware with different tag size distributions.

8.3 Future Work

- Production traffic validation: Deploying vastlint on live SSP or SSAI traffic to measure real-world error distributions, validate the synthetic corpus assumptions, and quantify actual revenue recovery. This is the highest-priority follow-up to the present work.
- Wrapper chain validation: Following redirect chains across multiple VAST wrapper hops and validating consistency across levels.
- SSAI-specific rule set: Rules specific to stitcher constraints, including transcoding format compatibility and bitrate requirements for specific delivery networks.
- Protocol-level integration: A reference implementation of VAST validation as a Prebid Server module, enabling standardized creative quality enforcement at the exchange layer.

9. Conclusion

A 17 KB CTV tag validates in 350 microseconds per bid -- 0.35 percent of a 100 ms bid window. A 44 KB tag validates in 2.1 milliseconds, within the creative selection budget. On 10 cores, the system sustains 16,236 tags per second, serving a mid-tier exchange from a single instance.

Inline VAST validation is not blocked by a technical constraint. The compute cost is small relative to the revenue risk. XSD schema validation, the nearest existing baseline, is faster but catches only

structural conformance issues -- element ordering and type constraints -- that do not correspond to production playback failures. The IAB's own VAST 4.2 XSD cannot validate a production tag without namespace modifications, and when it does validate, it rejects spec-compliant VAST macros as invalid URIs. What has kept meaningful validation out of pipelines is the deployment model: validation tooling has historically been built as offline, manual, web-based tools rather than embeddable libraries designed to run on the serving path.

vastlint is an open-source implementation of 108 VAST validation rules across six VAST versions, available as a Rust library, Go bindings, WebAssembly package, REST API, CLI, and VS Code extension. It is built to embed at the points in the pipeline where catching a bad tag changes the outcome: before the bid response goes out, before the creative enters rotation, before the tag gets stitched into the stream.

Source, benchmarks, corpus scripts, and integration documentation are at github.com/aleksUIX/vastlint.

References

[1] Google LLC. Google Authorized Buyers Real-Time Bidding Peer Guide. developers.google.com/authorized-buyers/rtb/peer-guide. States BidRequest.tmax "typically ranges from 80 to 1000 ms" and requires 85 percent of responses within deadline.

[2] IAB Tech Lab. OpenRTB 2.6 Specification. github.com/InteractiveAdvertisingBureau/openrtb2.x/blob/main/2.6.md. Defines tmax as "maximum time in milliseconds the exchange allows for bids to be received including Internet latency." Example 4 (video) uses tmax: 120.

[3] IAB Tech Lab. Video Ad Serving Template (VAST) 4.2. iabtechlab.com/standards/vast/. XSD schemas available for VAST 2.0.1 through 4.2. Prose specification for VAST 4.3.

[4] Penthera. The Biggest Advertising Challenges in VOD. penthera.com/post/the-biggest-advertising-challenges-in-vod. 2023. Reports VOD ad failure rates up to 40 percent.

[5] Google LLC. Troubleshoot video ad discrepancies. support.google.com/admanager/answer/4442429. Documents a 25 percent gap between served codes and ad impressions as a systemic condition.

[6] Berners-Lee, T., Fielding, R., and Masinter, L. RFC 3986: Uniform Resource Identifier (URI): Generic Syntax. IETF, 2005. tools.ietf.org/html/rfc3986.

[7] IANA. IANA Media Types Registry. iana.org/assignments/media-types/. Used for MIME type validation on VAST MediaFile and resource elements.

[8] ISO. ISO 4217: Currency Codes. iso.org/iso-4217-currency-codes.html. Used for Pricing currency attribute validation.

[9] Ad-ID LLC. Ad-ID Registry. ad-id.org. Used for UniversalAdId format validation.

[10] AdtechSchool. Real-Time Bidding: The Complete Technical Guide. 2025. Documents 150 ms

end-to-end auction timeline and DSP decision pipeline stages.

[11] Prebid.org. Prebid Server /openrtb2/auction endpoint. docs.prebid.org/prebid-server/endpoints/openrtb2/pbs-endpoint-auction.html. Minimum bidder tmax 50 ms (default), recommended 150 ms or more.

[12] Wikipedia contributors. Real-time bidding. en.wikipedia.org/wiki/Real-time_bidding. Notes DSPs must "determine the value of an individual impression in real time (less than 100 milliseconds)."

[13] Yuan, S., Wang, J., and Zhao, X. Real-time Bidding for Online Advertising: Measurement and Analysis. In Proceedings of the 7th International Workshop on Data Mining for Online Advertising (ADKDD '13), ACM, 2013. arXiv:1306.6542. Empirical analysis of a production ad exchange including bid latency distributions and auction mechanics.

[14] Murata, M., Lee, D., Mani, M., and Kawaguchi, K. Taxonomy of XML Schema Languages using Formal Language Theory. ACM Transactions on Internet Technology, 5(4):660-704, 2005. doi:10.1145/1111627.1111631. Formal analysis of the expressiveness and limitations of XML schema languages including XSD, RELAX NG, and Schematron.

[15] Wang, J., Zhang, W., and Yuan, S. Display Advertising with Real-Time Bidding (RTB) and Behavioural Targeting. Foundations and Trends in Information Retrieval, 11(4-5):297-435, 2017. doi:10.1561/1500000049. Comprehensive survey of RTB systems covering auction design, bidding strategies, and pipeline architecture.

All benchmarks: Apple M4 (10-core, 16 GB RAM), April 2026. Corpus generation scripts and raw data at github.com/aleksUIX/vastlint/tree/main/scratch.