Prebid for CTV-OTT: Use Cases, Common Integration Architectures, Challenges, Solutions

Contributors: Alex Krassel, Scott Kay, Mateusz Adamek-Siwirykow

Last Updated: Sep 15, 2022

Table of Content:

1. Overview

This document has been put together as an output of the Prebid.org's CTV-OTT Taskforce's meetings that took place in 2021.

The goal of this document is to express the collective knowledge that was assembled during these meetings as a single "Prebid in the CTV-OTT context manifesto" whitepaper that explains integration options and strategies, as well as challenges and possible solutions.

This document is intended for readers that have interest in Prebid-based solutions in the CTV-OTT context and want to learn all that is known on this subject. This includes publishers/broadcasters on the sell-side, advertisers/bidders in the buy-side, and ad tech providers in the middle.

It is assumed that the reader already has a general understanding of how Prebid-based solutions work in general, including the key/value pair based "buckets" that need to be pre-configured in the Ad Server.

Overall, Prebid in the CTV-OTT context focuses specifically on long-form video ad insertion, which adds a significant level of complexity compared to the display, native and short form video Prebid ads. While conceptually similar, long-form Prebid video ads add the notion of competitive exclusion within an ad pod and across multiple ad pods that need to factor in both directly sold and programmatic ads coming from disparate demand sources. It can also include additional actionable information about ad's position within an ad pod and/or across multiple ad pods. The ad duration is another additional dimension since the bids need to be normalized across ads of different duration (for example inserting two 15 second ads or a single 30 second ad into a pod).

To integrate with ad servers, Prebid for CTV-OTT uses the Prebid Server (or sometimes Prebid.js) at its core, with the same methodology of targeting pre-created line items in the Ad Server using the key/value targeting mechanism where the values signify the Prebid "buckets" of price ranges combined with other actionable and "bucketable" attributes (like video ad duration, competitive category, etc.).

It is also worth mentioning that many of the challenges covered in this document concentrate on the sell-side aspects of the Prebid technology integrating, since this is where multiple Prebid centric or Prebid aware components come together and need to be integrated. The buy-side is often isolated from these complexities by OpenRTB-based Prebid Server adapters. We will, however, call out the sell-side and buy-side differences as we cover individual challenges and solutions.

2. Known Use Cases and Integration Architectures

All the use cases listed below concentrate on the sell-side integration. The buy-side is represented by the Demand Sources concept, and a description of the inner workings of such demand sources are beyond the scope of this document.

Conceptually there are 4 ways to integrate Prebid in a CTV-OTT monetization solution:

1. Client-side using Prebid.js

2. Server-side using an SSAI server that calls the Prebid Server first in two flavors:
    a. SSAI server calls the Prebid Server first, then the Ad Server.

    b. SSAI server calls the Prebid Server based solution which acts as a proxy to the Ad Server and calls the Ad Server behind the scenes.

3. Server-Side using an SSAI server that calls the Ad Server first (“postbid”)

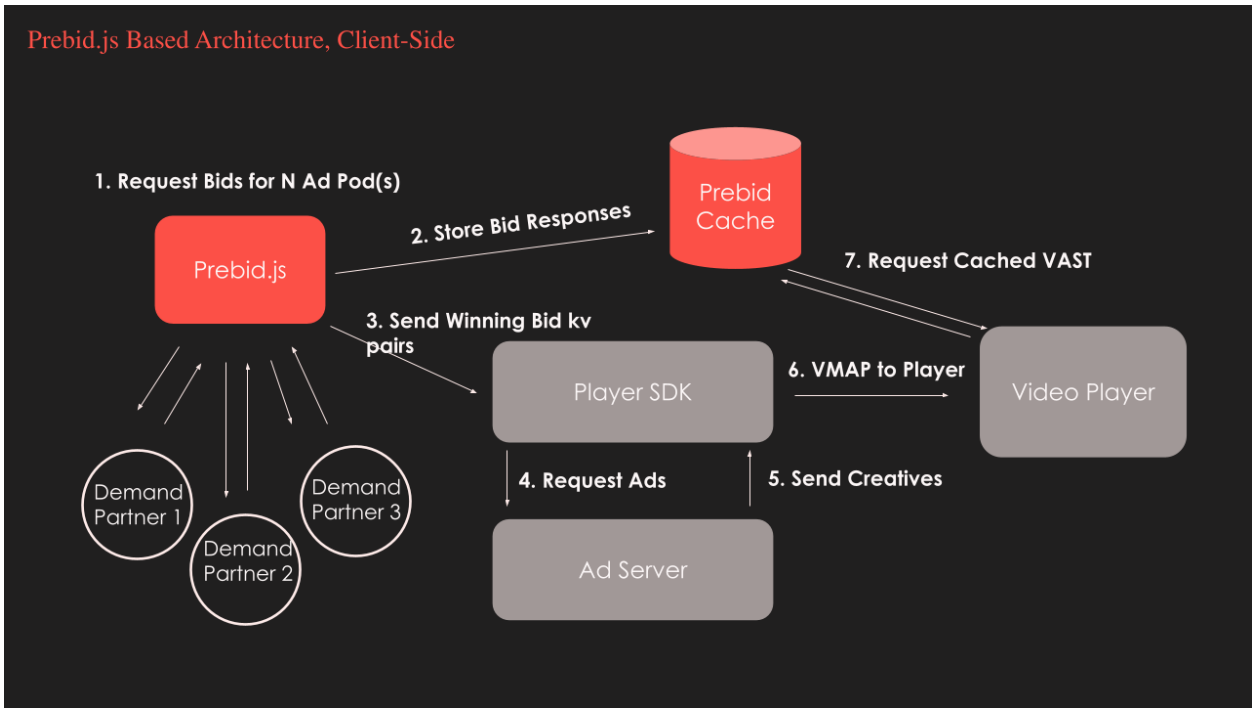
4. Server-Side using an SSAI server with Prebid Server based solution only (no Ad Server)

Each of these integration strategies have advantages and disadvantages. Below are the detailed descriptions for each of these integration strategies with all known pros and cons listed and explained.

2.1 Client-Side via Prebid.js

While not common for CTV because of the JavaScript implementation, this client-side method of integration is sometimes used with OTT solutions that target PCs, tablets and mobile web. This is also a common “entry level” integration methodology for publishers that want to experiment with a Prebid-based solution before investing heavily into any of the server-side based approaches.

The following diagram illustrates the general architecture in this use case:



This architecture requires integration of Prebid.js into the video player SDK on the publisher’s page that controls ad insertion on the client side.

The integration in this use case works as follows:

1. When the video player SDK is preparing to insert ads into client side video playback, it uses Prebid.js code integrated with the web page to request multiple Prebid ads to be inserted into the ad pod(s). There is no advance knowledge of where the Prebid ads will be inserted - this determination will be made later by the Ad Server. Prebid.js sends ad requests to multiple demand sources in parallel using preconfigured adapter modules, then picks the winning bids within each competitive category (this is done using optional Prebid.js’ deduplication logic which needs to be turned on for the OTT use case).

2. Prebid.js places all the VAST XML tags that survive deduplication into the Prebid Cache, and returns an object that contains key/value pairs that map URLs in the Prebid Cache to line items preconfigured in the Ad Server for the bid price/category/duration/etc. “buckets”.

3. The video player SDK gets the object returned by Prebid.js and prepares the call to the Ad Server.

4. The video player SDK then calls the Ad Server and passes the object with deduplicated Prebid bids to it along with other usual ad server call parameters.

5. The Ad Server combines the directly sold ads with programmatic ads returned by the Prebid.js using the K/V pairs mapped to “buckets” line items in the request, and produces the final ad payload in the form of VMAP (Note: if ad pods are constructed one at a time for live streams, the Ad Server may alternatively return a single VAST with multiple ads in it using VAST sequence number mechanism).

6. The VMAP or VAST XML returned by the Ad Server is passed into the video player for ad insertion. The video player parses the VMAP or VAST XML returned by the Ad Server and inserts the ad pods into the video content instructed.

7. For ads obtained via the Prebid Server, the VAST XML metadata is retrieved from the Prebid Cache.

Note: It is possible in this use case to configure the Prebid.js-based integration implementation to connect to demand sources through a Prebid Server behind the scenes to take advantage of the server-side demand adapters instead of the client side demand adapters. This does not change this integration architecture otherwise.

Pros and Cons of this integration architecture:

Pros:

a. Relatively simple integration, no SSAI complexity.

b. Built in support for ensuring that all the programmatic demand is competitively separated (core Prebid.js functionality).

c. Programmatic ads can outbid directly sold if desired/allowed.

d. Potentially reduced latency as the programmatic VAST tags can be unwrapped in parallel with the main Ad Server's decisioning logic (this, however, is not a part of the open source Prebid Server or Prebid Cache implementation, so custom logic would be required).

Cons:

a. This is a client-side integration, so it needs to be implemented and maintained in the client-side app for every supported platform.

b. This is a JavaScript based implementation, so may need to be ported to native code for client-side platforms that don't support JavaScript or limit JavaScript functionality.

c. No competitive category information is available beforehand since the Prebid.js code is called before the Ad Server. As a result, there is no way to tell the demand sources which categories to exclude.

d. No pod location or ad position in the pod information is available beforehand since the Prebid.js code is called before the Ad Server, and the Ad Server decides on the pod structure. As such, there is no way to take advantage of the upcoming OpenRTB 2.6 to signal the ad pod information to the bidders.

e. Can only be implemented using the key/value based "buckets" line items in the Ad Server. This, while being pretty standard header bidding Ad Server integration, is quite cumbersome and labor intensive to set up.

f. Implementation of advanced frequency capping techniques across directly sold and multiple demand sources is not trivial.

g. Because the categories of the directly sold ads are not known, some/many of the programmatic ads returned by Prebid server may competitively clash with directly sold ads and get dropped.

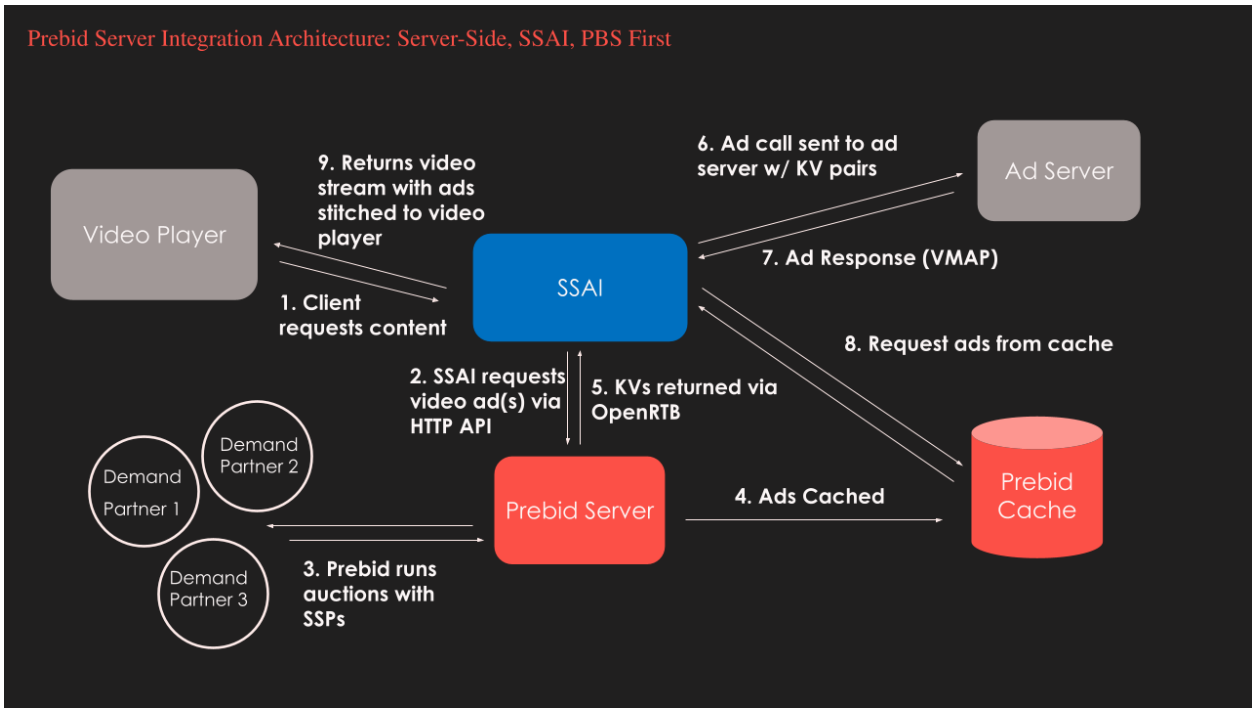
h. There can be instances of calling Prebid and the bidders where they are not authorized to deliver per Ad Server rules (for example being called when a Direct Sponsorship is already in place).

i. Difficult to implement dynamic floors for the inventory depending on yield management rules at the Ad Server level.

2.2.a Server-Side via SSAI Server, Prebid Server First

This is a fairly common method designed around the SSAI Server which directly integrates with the Prebid Server and Ad Server.

The following diagram illustrates the general architecture in this use case:



The integration in this use case works as follows:

1. A viewer using a client-side device (smart TV, Roku box, Fire TV device, Apple TV device, web browser running on a PC, tablet, or mobile phone, etc.) selects a video to watch or tunes into a live streaming channel. This results in the client side video player app making a request to the SSAI server for requested content.

2. The portion of the overall architecture that covers the content retrieval by the SSAI server is beyond the scope of this document, and is omitted in this diagram, as we only cover the ad insertion portion. Outside the scope of this document is also the mechanism by which the SSAI server determines the number of ads it needs/expects from all of the demand sources (both directly sold and programmatic). Once the SSAI server is ready to request the ad payload information, it sends an HTTP request to the Prebid Server using the standard PBS API.

3. The Prebid Server runs auctions in parallel against all preconfigured demand sources using appropriate demand source adapters. Prebid Server also performs necessary

deduplication logic on demand adapter responses (most commonly competitive category deduplication).

4. The Prebid Server places all the VAST XML tags that survive deduplication in the Prebid Cache, and returns a JSON structure that contains the key/values for the Prebid bids via OpenRTB back to the SSAI server. The keys are the category/pricing/duration/etc. “buckets”, and values are the URLs of the VAST tags in the Prebid Cache.

5. The SSAI server extracts the K/V pairs from the JSON structure returned by the Prebid Server and uses them to construct a request to the Ad Server.

6. The SSAI server then calls the Ad Server, passing the K/V pairs as part of the request string.

7. The Ad Server combines the directly sold ads with programmatic ads returned by the Prebid Server using the K/V pairs mapped to “buckets” line items in the request, and produces the final ad payload in the form of VMAP (Note: if ad pods are constructed one at a time for live streams, the Ad Server can return a single VAST with multiple ads in it using VAST sequence number mechanism).

8. As the SSAI server parses the VMAP or VAST XML returned by the Ad Server, it starts splicing the ads into the video stream as instructed. For ads obtained via the Prebid Server, the VAST XML metadata is retrieved from the Prebid Cache.

9. Finally, the complete video stream that contains both the main content and ad pods gets streamed to the video player running on the viewer’s CTV or OTT device.

Pros and Cons of this integration architecture:

Pros:

a. Integrates with the open-source version of the Prebid Server using standard protocols, no customization is necessary.

b. Built in support for ensuring that all the programmatic demand is competitively separated (core Prebid Server functionality).

c. Programmatic ads can outbid directly sold if desired/allowed.

d. Potentially reduced latency as the programmatic VAST tags can be unwrapped in parallel with the main Ad Server’s decisioning logic (this, however, is not a part of the open source Prebid Server or Prebid Cache Server implementation, so custom logic would be required).

e. No risk to the publishers as the Prebid Server calls are decoupled from the Ad Server, so the Prebid Server isn't a single point of failure.

Cons:

a. Requires supporting logic in the SSAI server (although this logic is quite generic regardless which vendor's Prebid Server is being used).

b. No competitive category information is available beforehand since the Prebid Server is called before the Ad Server. As a result, there is no way to tell the demand sources which categories to exclude.

c. No pod location or ad position in the pod information is available beforehand since the Prebid Server is called before the Ad Server, and the Ad Server decides on the pod structure. As such there is no way to take advantage of the upcoming OpenRTB 2.6 to signal ad pod information to the bidders.

d. Can only be implemented using the key/value based "buckets" line items in the Ad Server. This, while being pretty standard Prebid Ad Server integration, is quite cumbersome and labor intensive to set up.

e. Implementation of advanced frequency capping techniques across directly sold and multiple demand sources is not trivial.

f. Because the categories of the directly sold ads are not known, some/many of the programmatic ads returned by Prebid server may competitively clash with directly sold ads and get dropped.

g. There can be instances of calling Prebid and the bidders where they are not authorized to deliver per Ad Server rules (for example being called when a Direct Sponsorship is already in place).

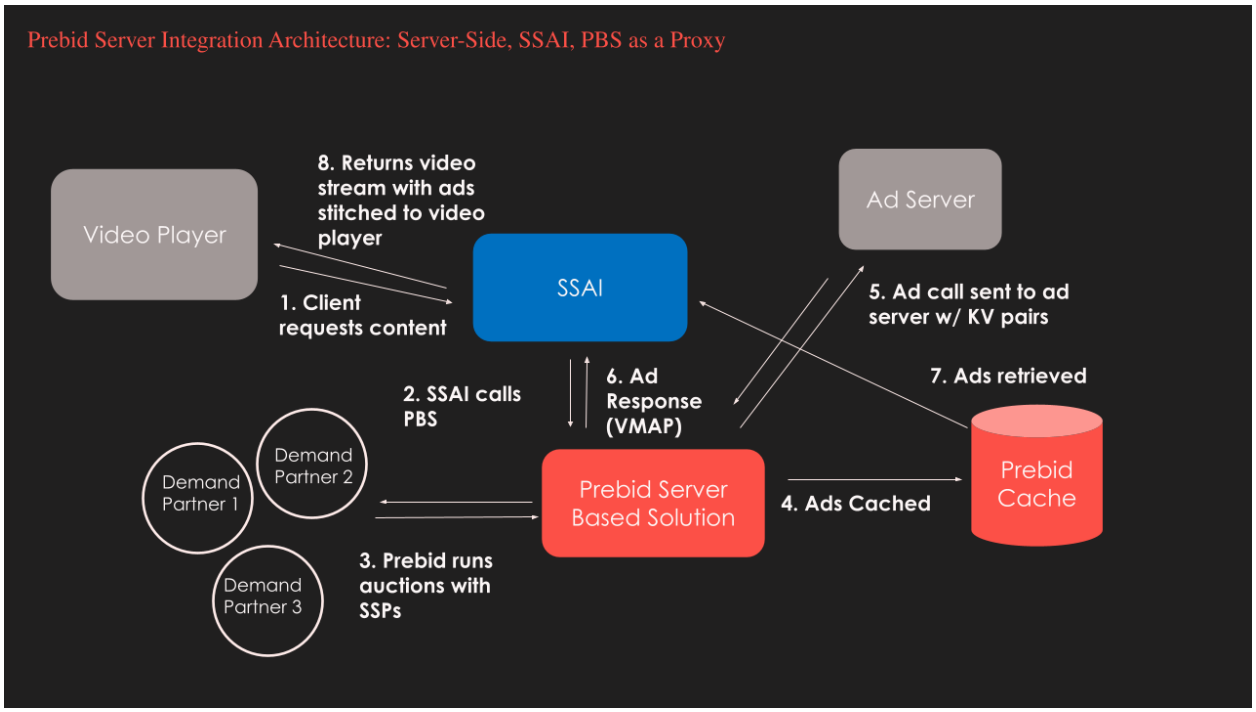
h. Difficult to implement dynamic floors for the inventory depending on yield management rules at the Ad Server level.

2.2.b Server-Side via SSAI Server, Prebid Server as a Proxy

This integration architecture is similar to the previous architecture. It is also designed around the SSAI server which calls the Prebid Server as if it was the Ad Server, with Prebid Server serving as a proxy and calling the Ad Server behind the scenes.

This goes beyond the open-source version of the Prebid Server, and requires a customized implementation that uses the open-source Prebid Server logic at its core, but also needs to implement a custom proxy and VMAP generation functionality.

The following diagram illustrates the general architecture in this use case:



The integration in this use case works as follows:

1. A viewer using a client-side device (smart TV, Roku box, Fire TV device, Apple TV device, web browser running on a PC, tablet, or mobile phone, etc.) selects a video to watch or tunes into a live streaming channel. This results in the client side video player app making a request to the SSAI server for requested content.

2. Same as in the previous use case, the content retrieval within the SSAI server is beyond the scope of this document. To get the ad payload the SSAI server calls the Prebid Server based solution.

3. The logic to determine the number of the ads needs to be implemented in the custom part of the Prebid Server based solution and passed to the open source based Prebid Server core. Prebid Server core runs auctions in parallel against all preconfigured demand sources using appropriate demand source adapters via OpenRTB communication protocol. Prebid Server core also performs necessary deduplication logic on demand adapter responses (most commonly competitive category deduplication).

4. The Prebid Server core places all the VAST XML tags that survive deduplication in the Prebid Cache, and returns a JSON structure that contains the key/values for the Prebid bids via OpenRTB back to the SSAI server. The keys are the category/pricing/duration/etc. “buckets”, and values are the URLs of the VAST tags in the Prebid Cache.

5. The logic implemented in the custom part of the Prebid Server based solution constructs the K/V pairs from the JSON structure returned by the Prebid Server core and uses them to construct a request to the Ad Server. The logic implemented in the custom part of the Prebid Server based solution then calls the Ad Server, passing in the key/value pairs as part of the request.

6. The Ad Server combines the directly sold ads with programmatic ads returned by the Prebid Server using the K/V pairs mapped to “buckets” in the request, and produces the final ad payload in the form of VMAP (Note: if ad pods are constructed one at a time for live streams, the Ad Server can return a single VAST with multiple ads in it using VAST sequence number mechanism). The logic implemented in the custom part of the Prebid Server based solution relays this VMAP (or VAST) to the SSAI server.

7. As the SSAI server parses the VMAP or VAST XML returned by the Prebid Server based solution, it starts splicing the ads into the video stream as instructed. For ads obtained via the Prebid Server, the VAST XML metadata is retrieved from the Prebid Cache.

8. Finally, the complete video stream that contains both the main content and ad pods gets streamed to the video player running on the viewer’s CTV or OTTdevice.

Note: Alternatively, instead of calling the Ad Server directly, the Prebid Server based solution can generate the finalized Ad Server invocation URL and return it back to the SSAI server as the HTTP 302 Redirect. This assumes that the specific SSAI server of choice can handle HTTP 302 in the response.

Pros and Cons of this integration architecture:

Pros:

a. Does not require custom integration with SSAI servers - SSAI server calls the Prebid Server based solution the same way it would call the Ad Server.

b. Built in support for ensuring that all the programmatic demand is competitively separated (core Prebid Server functionality).

c. Programmatic ads can outbid directly sold if desired/allowed.

d. Potentially reduced latency as the programmatic VAST tags can be unwrapped in parallel with the main Ad Server's decisioning logic (this, however, is not a part of the open source Prebid Server or Prebid Cache Server implementation, so custom logic would be required).

Cons:

a. Requires custom Prebid Server based solution.

b. The Prebid Server based solution can be the single point of failure for the entire content monetization, so publishers may be hesitant to use it.

c. No competitive category information is available beforehand since the Prebid Server is called before the Ad Server. As a result, there is no way to tell the demand sources which categories to exclude.

d. No pod location or ad position in the pod information is available beforehand since the Prebid Server is called before the Ad Server, and the Ad Server decides on the pod structure. As such there is no way to take advantage of the upcoming OpenRTB 2.6 to signal ad pod information to the bidders.

e. Can only be implemented using the key/value based "buckets" line items in the Ad Server. This, while being pretty standard header bidding Ad Server integration, is quite cumbersome and labor intensive to set up.

f. Implementation of advanced frequency capping techniques across directly sold and multiple demand sources is not trivial.

g. Because the categories of the directly sold ads are not known, some/many of the programmatic ads returned by Prebid Server may competitively clash with directly sold ads and get dropped.

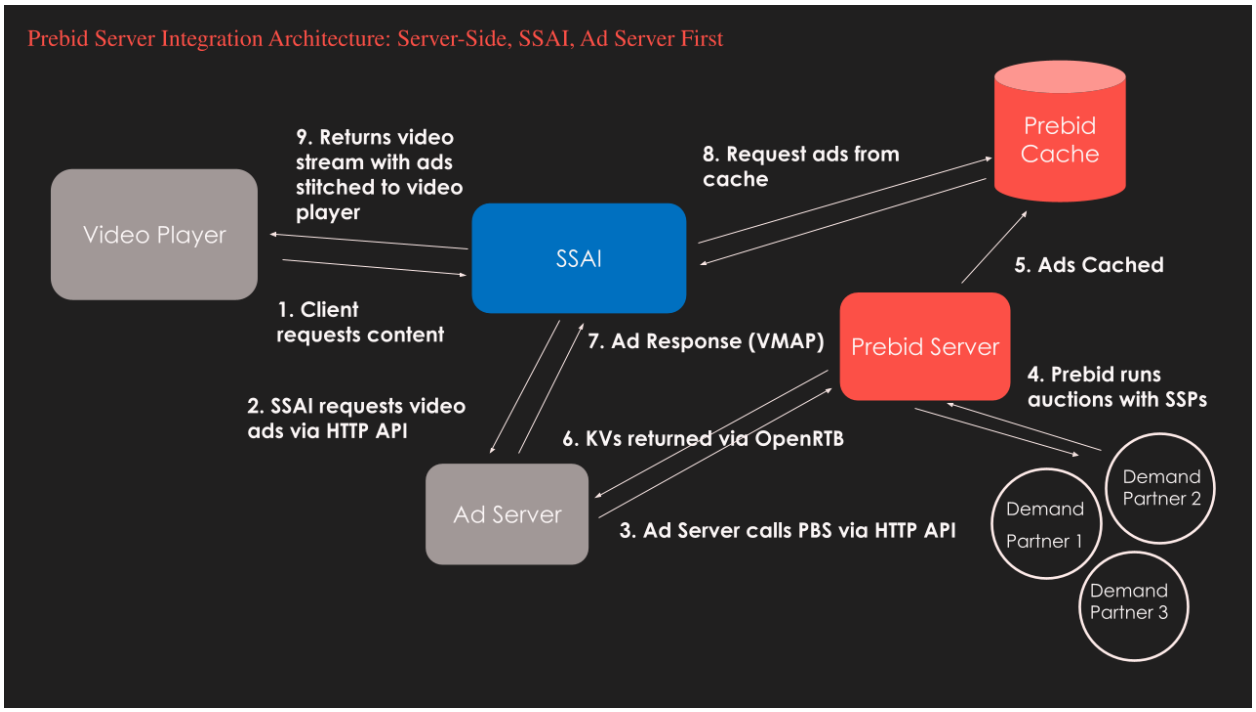
h. There can be instances of calling Prebid and the bidders where they are not authorized to deliver per Ad Server rules (for example being called when a Direct Sponsorship is already in place).

i. Difficult to implement dynamic floors for the inventory depending on yield management rules at the Ad Server level.

2.3 Server-Side via SSAI Server, Ad Server First

In this integration architecture, the SSAI server only communicates with the Ad Server, and the Ad Server calls the Prebid Server behind the scenes as the Ad Server is assembling the ad pods. This scenario is basically a “postbid” implementation, with the Ad Server using Prebid Server as a major conduit into various demand sources.

The following diagram illustrates the general architecture in this use case:



The integration in this use case works as follows:

1. A viewer using a client-side device (smart TV, Roku box, Fire TV device, Apple TV device, web browser running on a PC, tablet, or mobile phone, etc.) selects a video to watch or tunes into a live streaming channel. This results in the client side video player app making a request to the SSAI server for requested content.

2. Same as in the previous use cases, the content retrieval within the SSAI server is beyond the scope of this document. To get the ad payload the SSAI server calls the Ad Server in a usual manner.

3. The Ad Server internally implements custom logic to combine directly sold ads with programmatic ads obtained via the Prebid Server. The Ad Server calls the Prebid Server via standard Prebid Server HTTP API.

4. The Prebid Server runs auctions in parallel against all preconfigured demand sources using appropriate demand source adapters via OpenRTB communication protocol. Prebid Server also performs necessary deduplication logic on demand adapter responses (most commonly competitive category deduplication).

5. The Prebid Server places all the VAST XML tags that survive deduplication in the Prebid Cache, and returns a JSON structure that contains the key/values for the Prebid bids via OpenRTB back to the Ad Server. The keys are the category and pricing buckets, and values are the URLs of the VAST tags in the Prebid Cache.

6. The Ad Server utilizes custom logic to combine the internal directly sold ads with programmatic ads returned by the Prebid Server and constructs the resulting VMAP (or multi sequence number VAST if it's just a single pod).

7. The ad payload VMAP or VAST is returned back to the SSAI server.

8. As the SSAI server parses the VMAP or VAST XML returned by the Ad Server, it starts splicing the ads into the video stream as instructed. For ads obtained via the Prebid Server, the VAST XML metadata is retrieved from the Prebid Cache.

9. Finally, the complete video stream that contains both the main content and ad pods gets streamed to the video player running on the viewer's CTV or OTTdevice.

Note: The Ad Server may choose to embed the open source Prebid Server functionality directly, integrating much closer with the Prebid Server's logic and adapters. If the Prebid server logic is executed in parallel with the main Ad Server functionality, it may revert the "postbid" nature of this integration architecture and make it more like the typical "Prebid".

Pros and Cons of this integration architecture:

Pros:

a. Does not require custom integration with SSAI servers.

b. No need for key/value based “buckets” line items in the Ad Server.

c. Information about the pod number and position within pods is available as the call to the Prebid Server is made, so it can be signaled to bidders via OpenRTB 2.6.

d. Possibility to dynamically floor the inventory depending on yield management rules at the Ad Server level.

e. Better control of the Ad Server rules as applies to specific bidders (for example called a Direct Sponsorship is already in place).

Cons:

a. Proprietary “black box” closed-source integration logic between the Ad Server and Prebid Server.

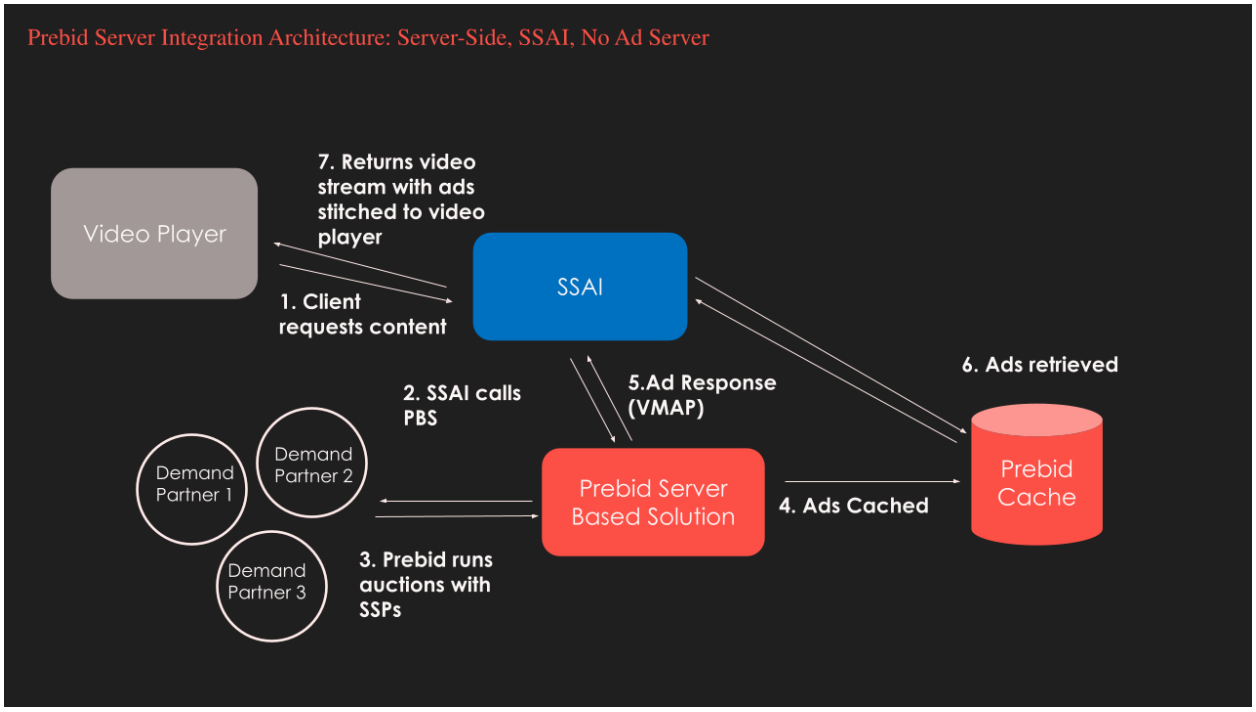
b. Potential increased latency due to the “postbid” nature of the solution (this may not be the case if the Prebid Server functionality is integrated directly into the Ad Server).

c. Potential waterfall implementations to handle competitive exclusion across programmatic demand sources (alternatively it may require more complex implementation to handle parallel Prebid Server calls).

2.4 Server-Side via SSAI Server, No Ad Server

In this integration method the SSAI server only communicates with Prebid Server for programmatic ads only. This use case also requires a custom Prebid Server based solution that goes beyond the standard open source Prebid Server functionality - it needs to provide ad podding planning logic as well as VMAP construction functionality.

The following diagram illustrates the general architecture in this use case:



The integration in this use case works as follows:

1. A viewer using a client-side device (smart TV, Roku box, Fire TV device, Apple TV device, web browser running on a PC, tablet, or mobile phone, etc.) selects a video to watch or tunes into a live streaming channel. This results in the client side video player app making a request to the SSAI server for requested content.

2. Same as in the previous use cases, the content retrieval within the SSAI server is beyond the scope of this document. To get the ad payload the SSAI server calls the Prebid Server based solution.

3. The logic to determine the number of the ads needs to be implemented in the custom part of the Prebid Server based solution and passed to the open source based Prebid Server core. Prebid Server core runs auctions in parallel against all preconfigured demand sources using appropriate demand source adapters via OpenRTB

communication protocol. Prebid Server core also performs necessary deduplication logic on demand adapter responses (most commonly competitive category deduplication).

4. The Prebid Server core places all the VAST XML tags that survive deduplication in the Prebid Cache, and returns a JSON structure that contains the key/values for the prebid bids via OpenRTB back to the Ad Server.

5. The custom portion of the Prebid Server based solution uses these key/value pairs to construct the VMAP (or VAST with multiple sequence numbers if only a single pod is requested), and returns the resulting VMAP or VAST tag to the SSAI server.

6. As the SSAI server parses the VMAP or VAST XML returned by the Prebid Server based solution, it starts splicing the ads into the video stream as instructed. The VAST XML metadata is for each individual ad is retrieved from the Prebid Cache.

7. Finally, the complete video stream that contains both the main content and ad pods gets streamed to the video player running on the viewer's CTV or OTT device.

Note: The VMAP / multi ad VAST tag building logic is fairly straightforward and generic. It may be eventually provided by the open source version of the Prebid Server, thus eliminating the need for custom logic.

Pros and Cons of this integration architecture:

Pros:

a. This is by far the simplest solution, and it does not require additional SSAI server integration - the SSAI server calls the Prebid Server based solution the way it would normally call an Ad Server.

b. Solves a lot of complexities of Prebid server / ad server integration, simplifies competitive separation and ad pod position signaling complexities, frequency capping, etc.

c. Eliminates the need for key/value based “buckets” line items complexity.

d. May end up being the preferred option in the longer run as programmatic becomes the predominant source of demand the way it played out in display (opinion expressed by some of the CTV-OTT taskforce participants).

Cons:

a. Requires custom Prebid Server based solution built around the Prebid Server core (this may go away if/when the open source version of the Prebid Server implements this functionality).

b. Lack of advanced features offered by ad servers, including dynamic floors for the inventory depending on yield management rules, etc.

c. Lack of robust reporting around Ad KPIs.

d. No support for directly sold ads.

3. Known Challenges and Possible Solutions

While the Prebid technology confers multiple significant benefits to publishers and advertisers alike, Prebid.org's CTV-OTT taskforce identified a number of technical challenges companies face when integrating Prebid technologies.

The taskforce ran a survey to identify the relative importance of all the identified challenges. The following section covers each of the identified challenges along with potential solutions in the order of relative importance, starting with the most significant items.

3.1 Integration With Ad Servers When The Prebid Server Is Called First

When Prebid technology is used for display, native, or short-form video ads, it already requires a fairly complex setup in the Ad Server to support it. Because most Ad Servers do not provide Prebid-specific APIs, the Prebid integration is typically implemented through the Ad Servers' key/value targeting functionality where the Prebid bid prices are rounded to the preconfigured values and expressed as the specific key/value based line items that are mapped to the CPMs in the Ad Server (often referred to as the "price buckets"). This often results in dozens of "buckets" / line items that need to be set up in the Ad Server (depending on the desired "price bucket" granularity).

Prebid for long-form video ads greatly complicates this methodology by adding additional dimensions such as competitive categories, and video duration. As a result, the Ad Server line item "buckets" for the long-form video end up being much more complex with a lot more permutations, which results in hundreds, thousands, or even tens of thousands of such line items that need to be set up in the Ad Server.

Most Ad Servers expose APIs that allow publishers to script such "bucket" / line item creation, however some Ad Servers severely limit access to such APIs to just their direct clients. This means that publishers often have to implement such complicated "buckets" (key/value based line items) on their own, and can't just use the prebuilt "bucket" creation scripts provided by their Prebid Server vendor. And Prebid Server vendors often lack access to such Ad Server APIs.

Advanced Ad Server API based scripts, however, do solve the challenges of the underlining Ad Server line item creation, and such scripts are relatively easy to develop.

Another important consideration highly relevant to use cases when the Prebid Server gets called first, before the Ad Server, is this: since it is the Ad Server that ends up constructing the VMAP, when the Prebid Server is called there is no information available of what directly sold ads will be inserted and in what position within the ad pods. As such, the Prebid Server can not pass the competitive exclusion categories to the external demand sources, so some of the bids

returned by the Prebid Server (via adapters) may clash competitively with the direct sold ads in the Ad Server, and potentially be discarded (depending how the direct sold ads are configured in the Ad Server).

Additionally, since there is no information about the structure of the pods yet, the Prebid Server can not use OpenRTB 2.6's signaling options to pass the pod positioning to the external demand sources. Nevertheless, the demand sources can use OpenRTB 2.6 flags in the responses and return multiple alternative bids for different ad pod positions as well as indicate a specific targeted pod position (using the "slotinpod" value). This information can be used as an additional dimension for the "buckets" (in addition to price, category, and video duration) to signal the desired ad pod positioning to the Ad Server.

To better understand the relevance of the above to a specific integration methodology, please refer to the earlier sections covering various integration architectures.

3.2 Deals Still Dominate

As of the time of this writing, deals are very important for the Prebid based long-form video ad insertion in the context of CTV-OTT. This is mostly due to the fact that fraudulent ad traffic is still very common on the open market for CTV-OTT (up to 25% quoted by some sources), so, as a result, deals are used for as much as 95% of the Prebid ads in the context of CTV-OTT.

The following are additional consideration related to the use of deals in CTV-OTT context:

- Deals describe relationships between publishers and SSPs, so the primary Ad Server may not be aware of deals.

- As such, deals are often defined in the 3rd party SSP systems.

- If needed, information can be incorporated into the price/category/duration “buckets” line items to make the Ad Server aware of deals, but it ends up increasing the number of “buckets” (line items) that need to be defined in the Ad Server, further exacerbating the complexity of the Prebid-based solution.

- When such deal information is incorporated into the Ad Server’s key/value based “buckets”, the Ad Server then can use it to prioritize deals.

- Separate deals are used for each Prebid ad independently of other ads in the pod.

- Current implementations of the Prebid Server are mostly passing the deal information from the OpenRTB based requests to the adapters/bidders without using this information internally. This could be an area of improvement for the Prebid Server.

- Proliferation of deals may require workflow complexity, and it can also greatly degrade system performance and introduce latency. As deals time out, the information about such deals needs to be removed from the “buckets” line item definitions if the deal information was included in such “buckets” definitions.

3.3 SSAI Server Integration Complexities

When the Prebid Server is called first by the SSAI server, it creates additional integration complexities as follows:

- The SSAI server needs to be aware of the Prebid Server, and explicitly call it first. While the call to the Prebid Server is simple, and parsing of the Prebid Server's response is fairly trivial, this also requires the SSAI server to implement some kind of a methodology to determine the number and/or total duration of ads it expects from the Prebid Server. The integration logic within the SSAI server does not need to be specific to a particular Prebid Server implementation though - once integrated with one vendor's Prebid Server it should work with any vendor's Prebid Server.

- The SSAI server needs to parse the Prebid Server's response and modify the call to the Ad Server to include the key/value pairs returned by the Prebid Server.

- Because the Prebid Server first integration architectures may introduce additional latency, it works best when the SSAI server implements some kind of a prefetch mechanism. This is especially important for the live streaming scenario. This way the SSAI server calls the Prebid Server and the Ad Server in advance over a period of a few minutes, which greatly reduces the load on all components involved in terms of QPS (queries per second).

- SSAI server needs to implement logic to gracefully fail a call to the Prebid Server if the Prebid Server does not respond within a predefined time. This is needed to protect against the situation where the Prebid Server fails, so the directly sold ads from the Ad Server are inserted even if the Prebid Server fails to respond.

- Generally speaking, the SSAI server is not expected to make any specific programmatic ad decisioning, but rather pass the ads returned by the Prebid Server to the Ad Server for the final ad insertion decisioning.

Additional general considerations:

- Since the SSAI server is the component that performs the actual ad insertion, it also ends up implementing the VMAP/VAST parsing as well as the VAST unwrapping when necessary. For integration scenarios where the Prebid Server is being called first, a custom Prebid Server based solution can take advantage of the fact that the Ad Server is called after the Prebid Server, and use the time it takes the Ad Server to respond to unwrap the VAST tags that the Pebid Server returned back to be passed into the Ad Server. This is also possible because the Prebid Server caches such ads in the Prebid Cache, so they can be unwrapped there.

- For VOD content all the entire ad payload needs to be determined before the content starts playing. This can result in Prebid ads being cached for quite some time in the Prebid Cache, so some ads can expire in the Prebid Cache before they can be inserted. This is usually less of a concern for live streaming, since in that case open ended manifests are being used, and ad pods are assembled and inserted one ad break at a time.

- The SSAI server is expected to act as a pass-through conduit for bids, including any of the identity / targeting information from the client app.

- It is up to the SSAI server / client app to implement an appropriate strategy for video ad tracker reporting (server side vs. client side). Tracker reporting approach has no bearing on the programmatic ad bids from the Prebid Server.

- In general, we need to promote non-proprietary open-source Prebid Server based SSAI solutions across the industry.

3.4 Identity and Device Type Challenges

Identity in the CTV space presents a challenge because there are significant differences between ecosystems created by different smart TV device manufacturers (for example Samsung vs. LG vs. Vizio vs. Roku, etc.).

Often two separate first party Ids are in play - one from the device manufacturer (for example Samsung now requires a Samsung account to finish setup of a new smart TV unit), while most CTV apps require a user login, so they have their own first party based Ids.

Id sync up is more complex because client side id sync is not an option for CTV. So only the server side option is available.

Some brands have access to TV and app manufactures' Id APIs, but not all.

OpenRTB extensions are often used in the Prebid Server for identity, mostly as pass-through to the adapters. Customers can set up permissions on such pass-through Id information per adapter (so it can be passed to some adapters but not all adapters if desired).

GDPR and other consent permissioning also needs to be passed in.

The Prebid Server is a fairly "dumb" proxy for identity pass-through - other than per-adapter filtering it doesn't do anything with the ids other than passing them through to the adapters to act upon.

Many SSAI servers pass through the ids, but many device manufacturers don't necessarily share them (like Samsung).

Contextual targeting can be used as an alternative way to match viewers with relevant ads, but it complicates frequency capping for the buyers / bidders because they don't know who the viewer is in this case. As such, any identifier (stable or semi-stable or session) could be better than nothing. These identifiers can be a "classic" device_id, an "Apple way" Identifier For Vendor - an identifier stable across a single device (or app on a device), but different across different apps on a device for example. Another option is a "session" identifier - semi-ephemeral identifier that allows capping across a session, but not across sessions.

This is another reason why deals are predominant in the CTV context - deals don't require user/device id but only an audience segment id.

Device type is more important for OTT vs. CTV (when the target device is not a smart TV), since for CTV the device type is typically a smart large screen TV device.

The Prebid Server gets only the user agent from the HTTP request, but providers like DeviceAtlas can be used to pinpoint the device based on the user agent. The open source

GoLang version of Prebid Server doesn’t currently hook into any device identification services, but it may provide the ability to hook into a device identification service of choice in the future.

There is currently no canonical device type nomenclature - this is a topic we can discuss with IAB in case they are interested in establishing the standards.

Some device makers have a lot of information in the user agent, others do not, so the application can be limited.

It would be nice to be able to pass additional device signals like the language (someone in the US for example can set up their device with a language other than English), potentially other device configuration details.

It would be also useful to have the device/app pass additional content related information - like the content type, content ratings, etc. IAB content taxonomy has hundreds of types that can be passed via OpenRTB (OpenRTB defines the content object).

3.5 Ad Break Bidding

As of the time of this writing OpenRTB 2.5 allows for requesting multiple video ads in a single call. Yet most systems ask for each ad in the pod individually instead of in a coordinated manner.

Even though each long-form video ad break contains a multi-ad pod, the open-source Prebid Server implementation breaks such ad pod requests into multiple uncoordinated single ad requests. This creates a lot of inefficiency because the demand sources often end up returning the same ad or an ad from the same competitive category, which leads to unnecessary deduplication and potentially suboptimal fill rate.

This is also due to the fact that currently many SSPs and DSP are not currently supporting multi-impression OpenRTB requests. As such, most systems can't ask for coordinated single pod-like requests. As noted earlier, when the Prebid Server is called first, there is no information about ad pods yet, the Prebid Server is merely asked to get a predefined number of ads. These ads may end up in different ad breaks and at different positions within the ad breaks - the specific ad break and position are assigned later by the primary Ad Server (FW/GAM/etc.). Nevertheless, asking for multiple ads at once would greatly improve the efficiency and fill rate.

The OpenRTB 2.6 specification has been released recently, and it adds the ability to pass the ad pod position information to provide more relevant information to bidders as well as the CPM per second (among a few other new attributes intended for the long form video). Currently it looks like OpenRTB 2.6 specification will be adopted by SSPs and DSPs in the second half of 2022.

3.6 Frequency Capping Across Programmatic and Directly Sold

Implementing a comprehensive frequency capping solution across both directly sold ads and Prebid/programmatic demand can be clunky and non-trivial.

In general, sell-side frequency tracking / capping functionality is provided by the primary Ad Server. Since the primary Ad Server makes the ad decisioning in Prebid for CTV-OTT use cases, and all the Prebid programmatic demand is being channeled through the primary Ad Server, it is up to the primary Ad Server to implement the frequency capping among all the line items (both directly sold and Prebid “buckets”).

Overall, IAB defines the frequency capping rules. Publisher side frequency capping can be viewer specific, country specific, contextual, etc.

Even though the primary Ad Server enforces the frequency capping rules for a publisher, the additional metadata returned from the Prebid demand sources can facilitate such frequency capping decisioning.

For example, the competitive category information is usually already included in the line item ”buckets”, so the Ad Server can factor it into its frequency capping strategy for ad categories. To go beyond that so brand level frequency capping could be enforced, additional information is required. This can theoretically be accomplished in the following 2 ways:

1. By including the brand information in the line item “buckets” - this, however, further exacerbates the complexity of maintaining thousands or even tens of thousands of line item “buckets” in the Ad Server.

2. By utilizing VAST ADID attribute - unfortunately less then 40% of programmatic VAST tags include this attribute at the time of this writing, so it would be unreliable as a general solution. Similarly, OpenRTB 2.3.1+ can include ADID in the response, but it is not a required parameter, so it doesn’t guarantee presence of the ADID information, and is rarely used.

In general, frequency capping needs to be applied intelligently by the publishers to achieve the right balance between viewer experience and monetization goals - with too much frequency capping the amount of programmatic demand retrieved via Prebid can diminish drastically. Ultimately, multiple publisher / bidder frequency capping relationship permutations are possible.

Depending on the specific implementation of the unified frequency capping across directly sold and programmatic demand sources may also run into privacy implications that need to be researched and addressed.

This also needs additional considerations to address the frequency capping needs / concerns of the buy-side.

3.7 Security and Fraud Prevention

Proliferation of fraudulent inventory is a significant problem for CTV in an open programmatic marketplace. Anecdotally, at the time of this writing, up to 25% of the CTV inventory used for the open programmatic market may be fraudulent. As a consequence, the majority of programmatic demand currently is served via deals, which solves the fraud problem, but significantly impedes the scalability of the Prebid based programmatic solutions due to complexity. Also proliferation of deals causes other problems - like significant computational burden, etc.

The Ads.Cert 2.0 suite of protocols, developed by the IAB, introduces authentication for open programmatic auctions and events which is aimed at significantly reducing the occurrence of CTV ad fraud. Prebid Server will implement Ads.Cert 2.0 Authenticated Connections and attach a cryptographically secure signature to all outgoing bid auctions to both identify the source of the bid request and protect against “man in the middle" attack vectors. SSAI servers can also use the Authenticated Connections protocol to sign impression notification events to be verified by the ad server to eliminate fraudulent signals.

The IAB Security Foundations Working Group is developing additional Ads.Cert 2.0 protocols to provide secure methods of verifying all parties of the supply chain delivery (Authenticated Delivery) and to verify the authenticity of end user devices (Authenticated Devices). The Authenticated Devices protocol in particular has a lot of potential to combat fraudulent CTV device traffic, such as emulated devices.

3.8 Google DAI/IMA Integration

For a long time Google appeared to be uninterested in directly supporting Prebid in GAM and Dynamic Ad Insertion (DAI). Direct DAI server - Prebid Server integration appears to be a non-starter.

A client side Prebid.js integration may be a viable alternative in this case. However, Google Interactive Media Ads (IMA) currently supports only a single call to the ad server (and the Prebid Server as a consequence) for the entire length of a live video stream, so a continuous live stream eventually runs out of ads (both directly sold and programmatic).

This issue is more on the IMA side, but many publishers use the IMA/DAI/GAM combo as they are usually marketed to publishers as a package. Many publishers use IMA because it greatly simplifies the client side functionally on many platforms.

At the time of this writing it is still unclear how important this issue is for the Prebid in the context of CTV-OTT, since FreeWheel ad server is much more dominant with long-form video content publishers. If the percentage of use cases this affects turns out to be low, it may be of a lower importance for this.

Also, the DAI server / Prebid Server integration challenges can be theoretically worked around by using the proxy solution that makes the calls to the Prebid Server instead of the primary ad server (see 2.2.b Server-Side via SSAI Server, Prebid Server as a Proxy). The call to such a proxy looks like a single ad server call to the DAI server.

In general, all integration architectures where the DAI ends up calling only a single entity (Prebid Server as a proxy, Ad Server first, or Prebid Server only) should work.

It is worth mentioning that as of the time of this writing, GAM is building a bridge to Prebid.js to better support javascript header bidding. The solution is currently in closed beta. Through this new bridge, publishers will be able to manage their Prebid relationships with the same user interface as Open Bidding, known as “yield groups.” They will also no longer need all the custom “buckets” line items, which will greatly simplify integrations.

3.9 Google Ad Manager

It appears at the time of this writing the majority of the CTV-OTT publishers use FreeWheel as their main Ad Server, so most emphasis is on FW. GAM is less researched/supported.

GAM does, however, provide an API that allows scripts being written that can automate “bucket” line item creation in a manner similar to FreeWheel’s API. Unlike FreeWheel’s API, GAM API is public with plenty of information available online.