PARAKEET Client-Side Decision Logic

A steppingstone towards MACAW functionality

Objectives

There are issues open against PARAKEET identifying gaps in SSP and DSP control. A few of those issues are listed here:

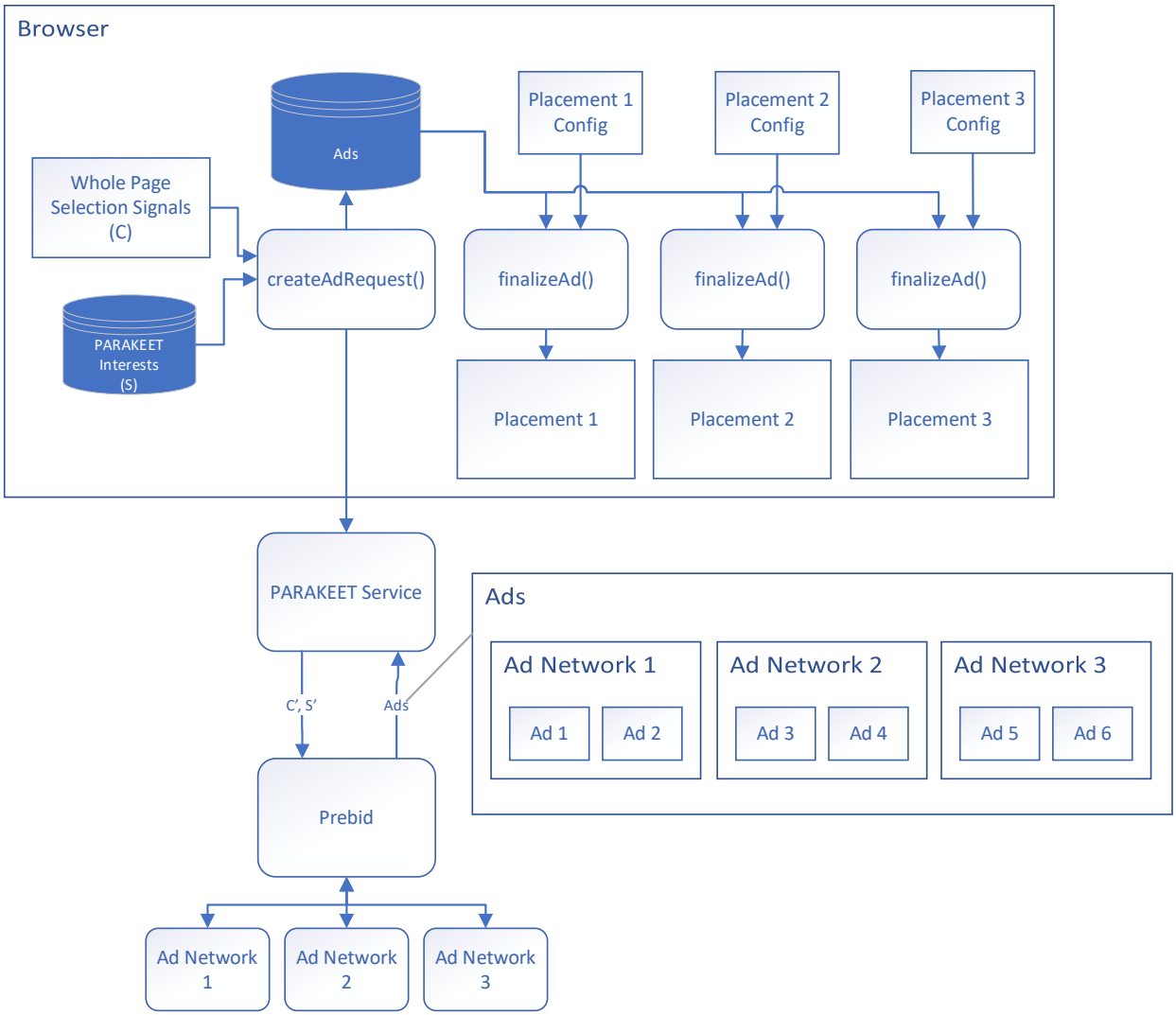
- Brand Safety
- Market Compliance
- Direct Sold Ads
- Frequency Capping
- Ad Quality

These are all important functionalities, and some are impacted due to signal coarsening (consider the challenge of handling market compliance with a fuzzy geolocation, or age restrictions with noised user information). MACAW will give ad servers the tools to address these issues by providing access to raw contextual page signals via MPC. To enable similar control in pre-MACAW PARAKEET, we will provide the capability to run sandboxed client-side logic with access to raw contextual page signals and user interests.

Contents

Overview and Diagram

In this model createAdRequest() will trigger a PARAKEET ad request. This call will retrieve a set of ads to be used in subsequent calls to finalizeAd(), which will be used to select an appropriate ad for each placement.



Example PARAKEET + Client-Side Logic Flow

Code

```
const adRequestSignals = {
  'adRequestUrl': 'https://prebid.example',
  'adProperties': [{
      'orientation': 'landscape', ...
    },{
      'orientation': 'portrait', ...
    }
  ],
  'publisherCode':'10931',
  'publisherAdUnit': 'publisher_ad_location_1',
  'targeting': {
    'interests': ['music', 'sports'],
    'geolocation': { 'lat': 41.5, 'lon': -81.7}
  },
  'anonymizedProxiedSignals': ['coarse-geolocation',
        'coarse-ua', 'targeting', 'user-ad-interests']
};

const myAuctionConfigForPlacement1 = {
  // Unencrypted client scoring logic
  'scoringLogicUrl': ...,
  // Publisher context accessible to bidding and scoring functions
  'auctionSignals': {...},
  // Context accessible to the scoring function
  'sellerSignals': {...}
};

const myAuctionConfigForPlacement2 = {...};
navigator.createAdRequest(adRequestSignals).then(adResponse => {
  navigator.finalizeAd(adResponse, myAuctionConfigForPlacement1)
    .then(opaqueCreativeUri =>
      { /* Create a fenced frame and set src to opaqueCreativeUri */ }
    ).catch(error => { /* Display fallback / direct sold creative */ });

  navigator.finalizeAd(adResponse, myAuctionConfigForPlacement2)
    .then(opaqueCreativeUri =>
      { /* Create a fenced frame and set src to opaqueCreativeUri */ }
    ).catch(error => { /* Display fallback / direct sold creative */ }
    );
})
```

Create Ad Request

As a first step, the publisher calls the `createAdRequest()` API to trigger the PARAKEET ad request. The request includes privatized interests and anonymized page context which ad providers use to identify relevant ads which are returned to the page.

Decision Logic

For each ad placement, the publisher calls `finalizeAd()` which first executes each ad network's `generateBid()` function on its set of returned ads.

```
generateBid(adResponse, auctionSignals, userInterests, browserSignals) {
  // Filter ads for placement eligibility, user optouts, or policy restrictions
  // Check browserSignals.prevwins for frequency capping
  return {'ad': adObject, 'bid': bidValue, 'render': renderUrl};
}
```

- `adResponse`: The reader's ad response object. Contains ads and metadata.
- `auctionSignals`: Page context passed by the publisher through the call to finalizeAd.
- `userInterests`: The set of interests accessible to the reader.
- `browserSignals`: Constructed by the browser. Can contain historical information like previous ad wins to allow generateBid to perform frequency capping.

After `generateBid()` selects the top ad for each reader, `scoreAd()` is run on each of those ads.

```
scoreAd(adMetadata, bid, auctionConfig, creativeScoringSignals, browserSignals) {
  // Filter and score ads based on the publisher's criteria
  return desirabilityScoreForThisAd;
}
```

- `adMetadata`: The adObject returned by generateBid.
- `bid`: The bid returned by generateBid.
- `auctionConfig`: The full configuration object used in the call to finalizeAd. Contains auctionSignals and sellerSignals.
- `creativeScoringSignals`: Metadata associated with the ad's render URL. Selected from the creativeScoringSignals field in the PARAKEET response.
- `browserSignals`: Constructed by the browser. Contains information the publisher may want to verify.

The ad with the highest positive score is selected for rendering.

Modified PARAKEET Response Format

PARAKEET's response format is modified to support the bidding and decision logic worklets that are executed for each placement. It consists of a set of ads for each participating ad network.

```
{
    // A list of ad responses from every participating DSP
    "winningAds": [
        {
            "reader": "dsp1.example",
            // Unencrypted client bid logic
            "bidLogicUrl": "https://dsp1.example/bidding.js",
            // Future DSP MPC inference URL and model binary
            "bidInferenceUrl": "https://dsp1.example/mpc",
            "bidModelUrl": "https://dsp1.example/bidding.out",
            "ads": [
                {
                    // Default bid and score, replaced with output of MPC
                    // models if the MACAW flow is invoked
                    "bid": 0.04,
                    "score": 0.4,
                    "creativeBundleUrl": [
                        "https://dsp1.example/render/creativeid1"
                    ],
                    // Arbitrary information about the ad
                    "adMetadata": {
                        "width": 100,
                        "height": 100,
                        "adtype": "image/native"
                    }
                },
                {...}
            ],
            // Arbitrary information the DSP may want to use in generateBid
            "metadata": {
                "historicalPlacementPerformance": {
                    "placement1": ...,
                    "placement2": ...
                }
            }
        },
        {
            "reader": "dsp2.example",
            ...
        }
    ],
    // Future SSP MPC inference URL and scoring model binary
    "scoringInferenceUrl": "https://ssp.ad-network.example/mpc",
    "scoringModelUrl": "https://ssp.example/scoringmodel-structure-for-service.out",
    // Per-creative metadata utilized by the scoreAd function
    "creativeScoringSignals": {
        "https://dsp1.example/render/creativeid1": {...},
        "https://dsp1.example/render/creativeid2": {...}
    }
}
```

Privacy Considerations

Both FLEDGE and PARAKEET expose a 1-bit leak: whether an auction returns an ad or not. In PARAKEET, due to the service's request anonymization, the 1-bit leak does not pose a significant risk with a single placement.

Given N placements in PARAKEET, a malicious party could control which placements show ads and which placements don't show ads, turning the 1-bit leak into an N-bit leak, which a collaborating publisher and ad provider could use to join C with S', recovering a user's differentially privatized segments. Repeated joins could resolve a user's true centroid.

In FLEDGE today, the 1-bit leak can be used to build a reliable signal for cross site tracking: Utilizing the 1-bit leak to build a cross site tracker · Issue #211 · WICG/turtledove (github.com). Both FLEDGE and PARAKEET will need to find mitigations for this issue.

The most effective mitigation for this is to render direct-sold ads in fenced frames with the same privacy bounds as programmatic ads. This needs to be explored further designs.